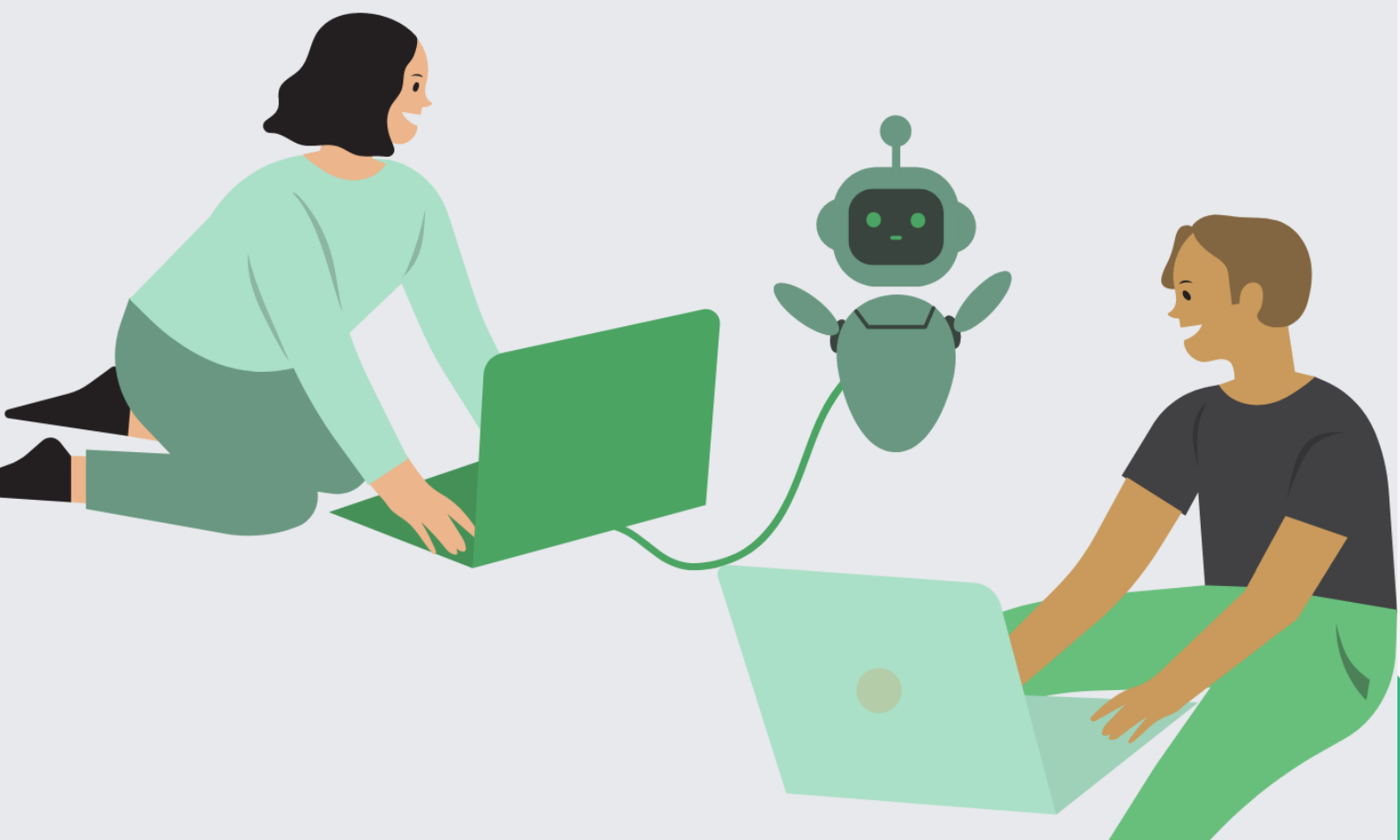




# CodER Modul o kodiranju i mikrokontrolerima: 20 sati lekcija



## Sadržaj

Uvod	4
1. Kontekst CodER projekta	5
1.1. Pogled na širu sliku: Gamifikacija i njezina uloga u obrazovanju	5
1.2. Cilj edukativnih metoda učenja pomoću igara u kodiranju i mikrokontrolerima	6
1.3. Publika i ciljna skupina ovih aktivnosti	8
DIO A: Modul CodER kodiranje (10 sati)	11
1. Uvod u programiranje	12
1.1. Što je programiranje	12
1.2. Zašto učiti programiranje? Koje su prednosti?	12
1.3. Proces razvoja programa i algoritmi	13
1.4. Programski jezici – Najtraženiji programski jezici	15
2. Postavljanje Pytona	16
2.1. Što je Python?	16
2.2. Najčešće korištena integrirana razvojna okruženja (IDEs) po industriji	17
2.3. Postavljanje Python integriranog razvojnog okruženja (IDE)	18
2.4. Pokretanje Pythona u naredbenom retku	23
3. Osnove programiranja u Pythonu	24
3.1. Osnove: Vrste podataka (osnovne i složene)	24
3.2. Varijable, izrazi i iskazi	25
3.3. Popisi, rječnici i tuplei	29
3.3.1. Popisi	30
3.3.2. Tuplei	35
3.3.3. Rječnici	39
3.4. Naredbe i petlje	43
3.4.1. Booleovski izrazi	44
3.4.2. Logički operatori	46
3.4.3. Ugnježđivanje if tvrdnje	46
3.4.4. Petlje	47
3.5. Razumijevanje tijeka izvršenja kroz funkcije	50
3.6. Shvaćanje sastavljanja dijelova – Kako napraviti program	54
3.7. Kako prilagoditi program svojim potrebama	57
3.8. Pogreške sintakse, vremena izvođenja i semantičke pogreške – rukovanje pogreškama u Pythonu	58



DIO B: MODUL CodER Mikrokontrolera (10 sati)	61
1. Uvod u mikrokontrolere	62
1.1. Što je mikrokontroler	62
1.2. Što je Arduino i njegove različite vrste	64
1.3. Koncepti: ulaz, izlaz, analogni, digitalni	68
2. Osnove programiranja s Arduino IDE	69
2.1. Postavljanje Arduino IDE i osnovnih naredbi	70
2.2. Programiranje u Arduino i učitavanje programa na ploču	73
2.3. Trepćući Led u Arduino	75
3. Aplikacije	78
3.1. Što je robotika?	78
3.2 Vrste robota	79
3.3 Upravljanje istosmjernim motorom sa štitom motora	84
3.4. Izrada paintbota upotrebom DC motora i Arduino	86
3.5 Izradite interaktivnu papirnatu igračku sa servo motorom	89
3.6. Brava vrata na daljinsko upravljanje	95
3.7. Mjerenje temperature, vlažnosti, svjetla i boje	101
3.7.1. Upotreba senzora u Arduino	102
3.7.2. Izradite teremin s Arduino i svjetlosnim senzorom	103
3.7.3. Roboti osjetljivi na boje	104
3.7.4. Uvod u DHT11 senzor	105
3.7.5. Izradite pametni ventilator za hlađenje	108



## Uvod

U suvremenom je svijetu sve veći naglasak na stjecanju digitalnih vještina. Međutim, trenutna neusklađenost ponude i potražnje na tržištu rada predstavlja nove izazove kako za poslodavce tako i za tražitelje posla. Ova neusklađenost prvenstveno pogađa mlade, budući da je nezaposlenost mladih jedan od najznačajnijih problema s kojima se Europa suočava. Projekt CodER predstavlja pokušaj premošćivanja ovog jaza materijaliziranjem znanja o kodiranju i mikrokontrolerima kroz inovativnu metodu sobe za bijeg (ERs) za mlade radnike i organizacije kako bi educirali mlade i povećali njihov interes i angažman za zanimanja koja su orijentirana na tehnologiju. Europski partneri koji sudjeluju u ovom projektu, financiranom iz Erasmus + programa su: Digijeunes (Francuska), Challedu (Grčka), Citizens in Power- C.I.P. (Cipar), RITE (Cipar), AKMI (Grčka) i Kalimera (Hrvatska).

Kroz ovaj će projekt, osobe koje rade s mladima i organizacije biti opremljeni novim vještinama i inovativnim metodama za privlačenje mladih ljudi, posebice mladih žena, na tehnološki orijentiranim putevima. Modul CodER predstavlja prvi korak prema realizaciji ciljeva ovog projekta namijenjenog široj javnosti. Ovaj modul omogućava osnovna znanja o programiranju i mikrokontrolerima temeljena na primjerima iz stvarnog života kako bi njegov sadržaj bio povezan sa svakodnevnom upotrebom. Pokušava usaditi logiku iza programiranja i mikrokontrolera kako bi između ostalih vještina promicao kultiviranje kritičkog mišljenja, kreativnost i vještine rješavanja problema.

Kontekst i pristup projekta CodER je prvenstveno predstavljen kako bi čitateljima pružio mogućnost razumijevanja razloga upotrebe pristupa učenja kroz igru/igranje. Prvi dio modula posvećen je razumijevanju upotrebe i prednosti programiranja, postavljanju Python integriranog razvojnog okruženja (engl. IDE) i prolasku kroz osnove programskog jezika Python za izradu malog programa. Drugi dio modula posvećen je upoznavanju mikrokontrolera i njihovoj upotrebi, postavljanju Arduino softvera i učenju kako ga koristiti za izvršavanje malih zadataka na mikrokontrolerima.



## 1. Kontekst CodER projekta

### 1.1. Pogled na širu sliku: Gamifikacija i njezina uloga u obrazovanju

Tijekom proteklog desetljeća došlo je do prijelaza s tradicionalnih metoda učenja usmjerenih na nastavnika na ono što se ponekad naziva pristupima obrazovanju usmjerenim na učenika. Dok tradicionalni pristupi obrazovanju uključuju "pasivni prijenos znanja s instruktora na učenika" (McGuire & Gubbins, 2010., str. 1), u pristupima obrazovanju usmjerenim na učenika, "od učenika se očekuje da postanu svjesni i vrednuju svoje vlastito iskustvo (...) gdje instruktor više nije baza informacija, već vodič koji sudjeluje u učenju" (McGuire & Gubbins, 2010., str.1). Ovaj suvremeni pristup metodama učenja obično je više tehnološki utemeljen i uključuje korištenje interaktivnih strategija kojima je cilj motivirati učenike da se aktivno uključe i sudjeluju u vlastitom procesu učenja.

Jedan od razloga za usvajanje ovih pristupa obrazovanju utemeljenih na tehnologiji, a posebno na STEM obrazovanju, jest taj što je značajan broj poslova sada usredotočen na aktivnosti i zadatke koji zahtijevaju korištenje interneta i digitalnih tehnologija (Daniel Calderón-Gómez et al. 2020. ), a u bliskoj budućnosti još više radnih mjesta morat će usvojiti ovu strukturu. To znači da će takvi poslovi zahtijevati stjecanje makar osnovnih digitalnih vještina, dok će povoljniji i visoko plaćeni poslovi zahtijevati još naprednije poznavanje digitalnih tehnologija (Karpinski et al., 2021.). Drugi čimbenik koji je igrao ulogu u kretanju prema pristupima obrazovanju usmjerenim na učenika, osim brzog tehnološkog napretka, je korelacija pronađena između neangažiranja, niske razine izvedbe i niske stope sudjelovanja u različitim obrazovnim okruženjima (kao što je navedeno u Levels et al., 2022. Callanan et al., 2009.). Kada je riječ o mladima, u dobi od 15 do 29 godina, to je dodatno istaknuto statističkim podacima dostupnim u Europi gdje 13,7% nije bilo u obrazovanju, zapošljavanju niti osposobljavanju (NEET-ovi) (Eurofound, 2022.). Stoga su se kreatori politika i organizacije mladih suočili s novim izazovima ponovnog podizanja interesa i reintegracije mladih u obrazovanje, zapošljavanje i/ili usavršavanje.

Na različitim razinama obrazovanja, tendencija pristupa učenju upotrebom elemenata učenja pomoću igara u kontekstu formalnog, neformalnog i informalnog obrazovanja od ranog djetinjstva do visokog obrazovanja i nadalje, postaje sve popularnija. U tom kontekstu pojavile su se definicije "gamifikacije" i "učenja pomoću igara". Iako se ova dva pojma često koriste naizmjenično, među njima postoje manje razlike. Pod pojmom "gamifikacija" obično podrazumijevamo "upotrebu elemenata dizajna igara u kontekstima koji se ne odnose na igre" (kao što je navedeno u Dichev & Dicheva, 2017., str. 2; Deterding, et al., 2011.; Hamari et al., 2014. ; Werbach, 2014.). U kontekstu obrazovanja, gamifikacija se odnosi na upotrebu „elemenata dizajna igara i doživljaja igara u dizajnu procesa učenja“ (Dichev & Dicheva, 2017,



str.2). Elementi dizajna igara sastoje se od dinamike, mehanike i komponenti igre (Dichev & Dicheva, 2017.). S druge strane, učenje pomoću igara (GBL) koristi tehnike koje primjenjuju dizajneri igara kako bi stvorili zabavno i impresivno iskustvo igranja za korisnika, s jedinim ciljem dizajniranja virtualnog okruženja za učenje koje uključuje svoje korisnike u obrazovne aktivnosti. Ova vrsta učenja može se odvijati i fizički i digitalno. Smatra se da učenje pomoću igara ima jasno definirane ishode učenja postignute kroz sadržaje igre (Sanchez, 2019.). Također se obično naziva ozbiljnim igrama, digitalnim učenjem ili obrazovnim igrama (Sanchez, 2019.).

Postoji mnoštvo dostupnih studija koje istražuju učinke učenja pomoću igara kod pojedinaca različite dobi, od učenika do studenata visokog obrazovanja i rjeđe kod odraslih (npr. Dichev & Dicheva, 2017; Ninaus et al., 2017; Sanchez et al., 2020.). Osnova za ovaj pokret koji ide u prilog učenju pomoću igara nalazi se u njegovom potencijalu da motivira učenike, posebno unutar dobnog raspona mladih, da se aktivno uključe i sudjeluju u vlastitom obrazovanju. Kako su opisali Dichev & Dicheva (2017), takvi pristupi učenju zahtijevaju "uranjanje na način sličan onome što se događa u igrama" (Dichev & Dicheva, 2017., str.2). Budući da su video igre prvenstveno dizajnirane za zabavu, "mogu proizvesti stanja željenog iskustva i motivirati korisnike da ostanu angažirani u pojedinoj aktivnosti" (Dichev & Dicheva, 2017., str. 12). Pokazalo se da je motivacija jedan od najvažnijih čimbenika za angažiranje učenika i uspješno obavljanje bilo koje aktivnosti koja zahtijeva vrijeme i trud. Kroz učenje pomoću igara, koncept vremena i truda pretvoren je u obrazovno i zabavno okruženje koje istovremeno omogućuje korisnicima nadogradnju znanja, vještina i stavova.

Jednostavan čin uokvirivanja aktivnosti ili zadatka kao igre, dovoljan je da dovede do pozitivnih psiholoških učinaka kao što su uključenost ili uživanje (Dichev & Dicheva, 2017.). Ovo gledište se, izgleda, temelji na pretpostavci da intrinzična motivacija proizlazi iz osnovne psihološke potrebe za zadovoljenjem tog "osjećaja kompetencije ili samoučinkovitosti" (Ninaus et al., 2017., str. 16) prisutnog u rješavanju problema ili u našem kontekstu, ispunjavanje cilja igre. Sposobnost igara da privuku pozornost igrača u stanje apsorpcije dok su isti angažirani u rješavanju problemskih zadataka, proizlazi iz njihove cilju usmjerene prirode koja igrača stavlja u stanje apsorpcije prilikom pokušaja izvršavanja ciljeve igre. Postoji pedagoška dimenzija svojstvena igrama koja se obično zanemaruje, kao što Tulloch (2014.) predlaže, naime „proces igara koji igrače podučava kako igrati“ (kao što je navedeno u Dichev & Dicheva, 2017., str. 23). Stoga metode učenja pomoću igara nastoje oponašati kako igra privlači igrače u stanje potpune apsorpcije i uključenosti kako bi se stvorilo jednako impresivno iskustvo u obrazovne svrhe.

## 1.2. Cilj edukativnih metoda učenja pomoću igara u kodiranju i mikrokontrolerima

Otkako je pandemija počela, naglasak stavljen na integraciju tehnologije u različite industrije postao je još izraženiji. Vizija Europske komisije predstavljena 2021. godine obuhvaća četiri



ključne točke koje će omogućiti digitalnu transformaciju Europe do 2030. godine. Jedna od tih kritičnih točaka je digitalna kompetencija potrebna za osiguranje razvoja otpornog europskog gospodarstva i društva. Cilj koji se treba postići do 2030. godine je da 80% ljudi posjeduje osnovne digitalne vještine (Europska komisija, 2021a).

Međutim, najveća prepreka ostvarenju europske digitalne transformacije je nedostatak digitalnih vještina u odnosu na njihovu veliku potražnju. Postotak poslodavaca koji se suočavaju s nedostatkom digitalno kompetentnih zaposlenika iznosi čak 70% (Europska komisija, 2021b). To povećava jaz u digitalnim vještinama koji postoji na tržištu rada, gdje čak i nisko kvalificirane do polukvalificirane profesije zahtijevaju određenu razinu digitalne kompetencije (Karpinski et al., 2021.). Kako je navedeno u Indeksu digitalne ekonomije i društva (DESI), 56% Europljana posjeduje osnovne digitalne vještine, ali samo 31% posjeduje digitalne vještine iznad osnovne razine (Europska komisija, 2021b). Stoga je važnost vještina rješavanja problema i računalnog razmišljanja visoko na dnevnom redu budući da samo 13% mladih i 6% odraslih posjeduje takve vještine u EU (Eurostat, 2020.).

Potrebne digitalne vještine prevedene su u "Okvir digitalnih kompetencija za građane" (DigComp) koji je razvila Europska komisija, koji uključuje niz vještina: 1) Informacijska i podatkovna pismenost, 2) Komunikacija i suradnja, 3) Kreiranje digitalnog sadržaja, 4) Sigurnost i 5) Rješavanje problema (Centeno et al., 2019.). Tvrdi se da će novi tehnološki zahtjevi zanimanja stvoriti novu podjelu unutar tržišta rada, koju će predstavljati tri kategorije digitalnih radnika: „digitalni prekarijat“ (s lošom ekonomskom situacijom); "tradicionalni digitalni rad" (uglavnom uključen u produktivne digitalne zadatke); i "inovativna klasa" (obavljanje produktivnih i komunikacijskih digitalnih zadataka)" (Calderón-Gómez et al., 2020., str. 7-8). Nadalje, predloženo je da će se "digitalni prekarijat" pretežno sastojati od mladih ljudi i žena (Calderón-Gómez et al., 2020., str. 9). U tom smislu, mladi koji su već u opasnosti od isključenosti iz društva i tržišta rada zbog svoje neaktivnosti ili nemogućnosti da nađu posao ili nastave daljnje obrazovanje i usavršavanja, nalaze se na raskrižju dvostrukog isključivanja.

Corneliussen (2021.) također naglašava kako nedostatak ženskih uzora u tim područjima i dominantna uloga muških ICT profesionalaca aktivno obeshrabruje mlade djevojke i žene od STEM orijentiranog puta. Općenito, povoljnije radne pozicije općenito više zauzimaju muškarci nego žene (Calderón-Gómez et al., 2020.) Kao što je već spomenuto, žene su zajedno s mladima najistaknutije skupine unutar digitalnog prekarijata. Istodobno, žene su „nedovoljno zastupljene u inovativnoj klasi, dok su također previše zastupljene u tradicionalnom analognom radu“ (Calderón-Gómez et al., 2020., str. 8). Studija pokazuje da muškarci češće nego žene, provode više vremena na internetu radeći aktivnosti vezane za posao, a među "onima sa sveučilišnom diplomom, 77,5% muškaraca u usporedbi sa 70,0% žena koristi internet na poslu" (Calderón-Gómez et al., 2020., str. 18-20). Nadalje, žene imaju tendenciju više koristiti digitalne tehnologije za komunikaciju i mobilnu upotrebu, dok muškarci više koriste digitalne tehnologije za produktivnu i uredsku upotrebu, te za višestruke



funkcije i namjene (Calderón-Gómez et al. 2020). Općenito, u usporedbi s muškarcima, žene rjeđe koriste internet i/ili digitalne tehnologije za složenije namjene.

Zbog svega navedenoga, kodiranje i mikrokontroleri su vrlo relevantni za trenutne potrebe tržišta rada. Kombinacija učenja pomoću igara u području kodiranja i mikrokontrolera omogućuje nam dosezanje naše ciljane skupine. Na temelju prednosti o kojima se raspravljalo u prethodnom poglavlju, učenje pomoću igara služi kao sredstvo za prijenos znanja i vještina kroz zabavno i smisleno iskustvo učenja. Točnije, studije koje su koristile Sobe za bijeg kao svoj alat podučavanja, pokazale su njihovu sposobnost stimuliranja pozitivnih emocija, povećanje razine angažmana i stvaranje sveukupnog pozitivnog iskustva učenja za učenike (Llerena-Izquierdo & Sherry, 2022.; Sánchez-Martín et al., 2020.). Ohrabrujući rezultati takvih studija predstavljaju prilike za projekt CodER u premošćivanju jaza u digitalnim vještinama i nude nove mogućnosti mladima koji su u opasnosti od isključenosti.

Iduća stavka koja je vrlo relevantna u našem nastojanju stvaranja digitalnih soba za bijeg je ograničenost fizičkih soba za bijeg. Kako Fotaris i Mastoras (2019., str. 3) ističu da "nije izvedivo ili legalno zaključati podskup razreda u prostoriju i čekati dok ne odgonetnu svoj izlaz iz nje (...), mnoge sobe za bijeg dizajnirane za učionicu su vraćene na grupnu stolnu aktivnost koja uključuje niz zaključanih kutija". Ali ovom rješenju nedostaje imerzivan karakter koji je tipičan za sobe za bijeg. Postoje i drugi izazovi koje je potrebno riješiti pri implementaciji soba za bijeg u obrazovne svrhe, uključujući posvećenost vremena zbog radno intenzivnog procesa potrebnog za stvaranje soba za bijeg, vremenska ograničenja koja ograničavaju testiranje igranja što može dovesti do još više problema kao što je loš dizajn i neuravnotežene poteškoće, te na kraju, velike grupe koje otežavaju način na koji će učenici sudjelovati u sobi za bijeg ili koliko će ona sama biti zanimljiva (Fotaris & Mastoras, 2019.). U tom pogledu, generator digitalne sobe za bijeg koji će biti stvoren kao krajnji rezultat ovog projekta rješava većinu ovih ograničenja i omogućuje osobama koje rade s mladima veću fleksibilnost u dizajniranju takvih prostora koji bi odgovarali potrebama njihovih učenika.

Štoviše, Llerena-Izquierdo & Sherry (2022.) ističu da su nedostatak vodiča za dizajn i iskustva korisnika dva ograničavajuća čimbenika za široku prilagodbu takvih alata u formalnom, neformalnom i informalnom obrazovnom kontekstu. Ovo je još jedna važna stavka koju pokušavamo riješiti stvaranjem ovog modula kao prvog koraka, te ostalim očekivanim projektnim rezultatima koji će omogućiti osobama koje rade s mladima potrebna znanja i vještine za usađivanje novih znanja mladima.

### 1.3. Publika i ciljna skupina ovih aktivnosti

Publika kojoj se projekt CodER obraća su članovi lokalnih, nacionalnih, europskih organizacija mladih. To uključuje sve vrste profesionalaca koji rade za organizacije mladih, koji se bave skupinama u nepovoljnom položaju kao što su NEET-ovci, osobe koje rano napuštaju





školovanje, migranti, izbjeglice i tražitelji azila i bivši zatvorenici. Također, organizacije mladih koje se fokusiraju na pružanje neformalnog obrazovanja vezanog uz tehnologiju i inovacije, organizacije mladih i/ili centri za mlade koji nastoje povećati zapošljivost mladih i organizacije mladih koje se bave STEM obrazovanjem, također su dio ciljne publike uz mlade radnike koji se bave programiranjem, mikrokontrolerima, edukativnim sobama za bijeg. Nadalje, ciljna publika CodER-a proteže se na IT stručnjake, predstavnike ICT startup-ova, znanstvenike, istraživače, sveučilišno osoblje, kreatore mišljenja i utjecajne multiplikatore kao što su udruge za kodiranje, inovacijski centri, predstavnici državnih tijela, malih i srednjih poduzeća, društvenih i sektorskih tijela, stručnjaci za marketing proizvoda i lokalni općinski zaposlenici.

Kroz ovu publiku želimo doprijeti do naše ciljne skupine, mladih ljudi, kako bismo im prenijeli znanje stečeno od strane omladinskih organizacija. Ciljna skupina uključuje mlade ljude koji pripadaju raseljenoj populaciji (migranti, tražitelji azila, izbjeglice, manjinske populacije), mlade ljude koji su općenito u opasnosti od socio-ekonomske isključenosti (rano napuštanje škole, NEET itd.) i mlade ljude s poremećajima u učenju. Kao što je već spomenuto, veliki broj mladih ljudi se bori s nezaposlenošću zbog nedostatka vještina kodiranja koje poslodavci traže kod svojih zaposlenika. Stoga je ovaj projekt osmišljen na način da će omladinske organizacije naučiti osnovama programiranja i mikrokontrolera, osigurati im metodološki i pedagoški vodič o tome kako koristiti sobe za bijeg u svrhu edukacije mladih kroz razigrano i zabavno iskustvo, kreirati niz različitih scenarija koje mogu koristiti kao predloške ili ih prilagoditi i stvoriti generator digitalne sobe za bijeg.

## Reference

Calderón-Gómez, D., Casas-Mas, B., Urraco-Solanilla, M., & Revilla, J. C. (2020). The labour digital divide: digital dimensions of labour market segmentation. *Work Organisation, Labour & Globalisation*, 14(2), 7-30.

Centeno, C., Vuorikari, R., Punie, Y., O'Á, W., Kluzer, S., Vitorica, A., ... & Bartolome, J. (2019). *Developing digital competence for employability: Engaging and supporting stakeholders with the use of DigComp* (No. JRC118711). Joint Research Centre.

Corneliussen, H. G. (2021). *Women empowering themselves to fit into ICT*. In *Technology and Women's Empowerment*. Taylor & Francis

Dichev, C. & Dicheva, D. (2017). Gamifying education: what is known, what is believed and what remains uncertain: a critical review. *International Journal of Educational Technology in Higher Education* 14, 9. <https://doi.org/10.1186/s41239-017-0042-5>

DigitalEurope (n.d.). Key indicators for a stronger digital Europe. <https://www.digitaleurope.org/key-indicators-for-a-stronger-digital-europe/>

European Commission (2021a). *Europe's Digital Decade: digital targets for 2030*. [https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/europes-digital-decade-digital-targets-2030\\_en](https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/europes-digital-decade-digital-targets-2030_en)

European Commission (2021b). *Digital skills and jobs*. <https://digital-strategy.ec.europa.eu/en/policies/digital-skills-and-jobs>



Eurofound (2022). NEETs. <https://www.eurofound.europa.eu/topic/neets>

Eurostat (2020). *Being young in Europe today – digital world*. [https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Being\\_young\\_in\\_Europe\\_today\\_-\\_digital\\_world&oldid=528990#Information\\_and\\_communications\\_technology\\_skills](https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Being_young_in_Europe_today_-_digital_world&oldid=528990#Information_and_communications_technology_skills)

Fotaris, P., & Mastoras, T. (2019). Escape rooms for learning: A systematic review. In *Proceedings of the European Conference on Games Based Learning*, 235-243.

Karpinski, Z., Biagi, F., & Di Pietro, G. (2021). Computational Thinking, Socioeconomic Gaps, and Policy Implications. IEA Compass: Briefs in Education. 12. *International Association for the Evaluation of Educational Achievement*

Levels, M., Brzinsky-Fay, C., Holmes, C., Jongbloed, J., & Taki, H. (2022). The dynamics of marginalized youth: *Not in education, employment, or training around the world*. Taylor & Francis.

Liu, Z.-Y., Shaikh, Z. A., & Gazizova, F. (2020). Using the Concept of Game-Based Learning in Education. *International Journal of Emerging Technologies in Learning (IJET)*, 15 (14), 53–64. <https://doi.org/10.3991/ijet.v15i14.14675>

Llerena-Izquierdo, J., & Sherry, L. L. (2022). Combining Escape Rooms and Google Forms to Reinforce Python Programming Learning. In Á. Rocha, P. C. López-López & J. P. Salgado-Guerrero (Eds.), *Communication, Smart Technologies and Innovation for Society* (pp. 107-116). Springer, Singapore.

Mcguire, D. & Gubbins, C. (2010). The Slow Death of Formal Learning: A Polemic. *Human Resource Development Review*. 9. 249-265. [10.1177/1534484310371444](https://doi.org/10.1177/1534484310371444).

Ninaus, M., Moeller, K., McMullen, J., & Kiili, K. (2017). Acceptance of Game-Based Learning and Intrinsic Motivation as Predictors for Learning Success and Flow Experience. *International Journal of Serious Games*, 4(3), 15–30.

Sanchez, E. (2019) Game-Based Learning. In: Tatnall A. (eds) *Encyclopedia of Education and Information Technologies*. Springer, Cham. [https://doi.org/10.1007/978-3-319-60013-0\\_39-1](https://doi.org/10.1007/978-3-319-60013-0_39-1)

Sánchez-Martín, J., Corrales-Serrano, M., Luque-Sendra, A., & Zamora-Polo, F. (2020). Exit for success. Gamifying science and technology for university students using escape-room. A preliminary approach. *Heliyon*, 6(7). <https://doi.org/10.1016/j.heliyon.2020.e04340>

Vasilescu, M. D., Serban, A. C., Dimian, G. C., Aceleanu, M. I., & Picatoste, X. (2020). Digital divide, skills and perceptions on digitalisation in the European Union—Towards a smart labour market. *PLoS ONE*, 15(4), 1–39. <https://doi.org/10.1371/journal.pone.0232032>



## DIO A: Modul CodER kodiranje (10 sati)

### Opis:

U ovom poglavlju se pruža opsežan uvod u programiranje uz njegovu upotrebu i primjenu na primjerima iz stvarnog života. Na samome početku su objašnjene prednosti programiranja i njegova primjena na tržištu rada, kao i proces nakon kojeg slijedi algoritam i razvoj programa. Zatim, drugo potpoglavlje objašnjava što je Python, korištenje integriranog razvojnog okruženja (IDE) i kako ga postaviti. Ostatak modula je posvećen različitim vrstama podataka, njihovoj upotrebi i primjeni kroz primjere, kao i razvoju kratkih programa.

### Trajanje rada:

10 sati

### Ishodi učenja:

Do kraja ovog tečaja polaznici će biti sposobni:

- ⇒ Prepoznati vrijednost i korištenje programiranja
- ⇒ Shvatiti tijek izvršenja u programima
- ⇒ Koristiti osnovnu sintaksu za pristup, izmjenu i brisanje različitih vrsta podataka u Pythonu
- ⇒ Koristiti Python za izradu malih programa

### Materijali i sredstva potrebni za rad:

- ⇒ Računalo ili prijenosno računalo
- ⇒ Pristup internetu
- ⇒ Spyder
- ⇒ Visual Studio Code

### Praktičnosti:

Za ovaj dio modula predviđeno je 10 sati učenja. Svaki dio modula ima određeno vrijeme, ali učenik ili nastavnik/trener može slobodno odlučiti koliko će potrošiti na svaku podtemu ovisno o predznanju i angažmanu u sličnim temama. Sadržaj se temelji na progresivnom razvoju i služi za pružanje osnovnih znanja i vještina početnicima.



## Potpoglavlja:

1. Uvod u programiranje
2. Postavljanje Pythona
3. Osnove programiranja u Pythonu

## 1. Uvod u programiranje

- ⇒ **Broj sudionika:** 1-10 po voditelju
- ⇒ **Trajanje:** 0.5-0.75 sati
- ⇒ **Oblici provođenja nastave:** predavanje, prezentacija
- ⇒ **Potrebni materijali:** prezentacija

### 1.1. Što je programiranje

Programiranje je proces stvaranja programa posvećenog izvršavanju određenog zadatka putem računala.

Programi uključuju postupak korak po korak, obično opisan kao stvaranje recepta, koji se koristi za komunikaciju s računalom korištenjem niza unaprijed definiranih sintaksi i funkcija.

Možda mislite da ovo zvuči pretjerano komplicirano, ali nije. To je kao učenje novog prirodnog jezika iz početka, morate razumjeti njegovu sintaksu i vokabular kako biste ga mogli pravilno koristiti. Jednako važan aspekt je vježbanje vlastitih vještina kako biste postali bolji u bilo kojem prirodnom ili programskom jeziku kojeg učite.

Prirodni jezik (Hrvatski)	Programski jezik
John čita svaki dan	Ispis ("Hello, World")

**Tablica 1** - Usporedba prirodnog i programskog jezika

### 1.2. Zašto učiti programiranje? Koje su prednosti?

Možda se pitate koja je svrha učenja programskog jezika i koje prednosti ono uključuje.

Kako tvrdi Steve Jobs: "Svatko u ovoj zemlji trebao bi naučiti programirati računalo jer vas ono uči razmišljati". Temeljeno na ovom citatu, jedna od glavnih prednosti programiranja je da razvija računalno razmišljanje. Računalno razmišljanje "odnosi se na nečiju sposobnost identificiranja, testiranja i primjene mogućih algoritamskih rješenja na trenutani problem i na analogne probleme koji bi se mogli pojaviti u novom kontekstu ili situaciji" (Karpinski et al., 2021., str. 3). Ovo se također izravno odnosi na vještine rješavanja problema, koje se smatraju



poželjnim i temeljnim vještinama koje moraju biti sposobne prilagoditi se promjenama i zahtjevima trenutnog tržišta rada (kao što je navedeno u Karpinski et al., 2021; Czaja i Urbaniec, 2019; Rakowska i Cichorzewska, 2016; Slavinskis et al., 2015). Vještine rješavanja problema visoko su cijenjene među poslodavcima i smatraju se jednom od najvažnijih mekih vještina 21. stoljeća.

Sposobnost korištenja programskog jezika ne znači da svatko treba postati programer, već da programiranje može biti korisno za svaku karijeru i može otvoriti nove putove u karijeri.

Fleksibilnost koju dopušta programiranje još je jedna od njegovih glavnih prednosti budući da će se pojaviti nove mogućnosti zapošljavanja koje su više usmjerene na kodiranje. Nadalje, osnovno znanje programiranja će biti potrebno za svaki posao u budućnosti. To povećava njegovu vrijednost kao temeljne vještine za sutrašnje društvo i tržište rada.

Svaka osoba koja osjeća da stagnira u svojoj karijeri može naučiti programski jezik kako bi povećala svoj prihod i izgled za karijeru, kao i dovela svoje vještine rješavanja problema korak dalje. Stoga je programiranje sveobuhvatna vještina koja zahtijeva kako tehničke tako i meke vještine i može je naučiti svatko tko je voljan isprobati.

### 1.3. Proces razvoja programa i algoritmi

Ako još uvijek čitate ovaj modul, dopustite nam da objasnimo što je to algoritam i kako se program razvija.

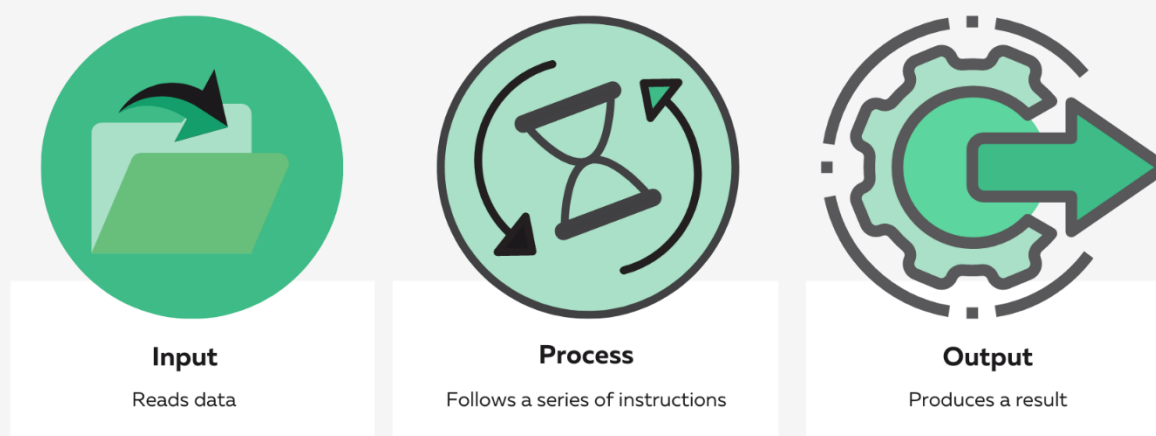
Algoritmi su sastavni dio programa jer predstavljaju korake koji su potrebni za dovršetak zadatka određenog kôdom. Dobar algoritam mora uključivati sljedeće: korake ili aktivnosti koje su potrebne za dovršetak zadatka, ispravan redoslijed tih aktivnosti i njihov završetak. Čest primjer koji se koristi za razumijevanje procesa algoritma je recept. Sljedeći recept "Ispecite tortu" može se prevesti u algoritam:



Slika 1 - Vizualni prikaz algoritma

Kao što možete vidjeti, redoslijed svake aktivnosti ili koraka temelji se na logičnom redoslijedu. Na primjer, ne možete staviti posudu u pećnicu, a da prethodno ne stavite sastojke u posudu; ne ako želite imati plodonosan postupak. Isto vrijedi i za algoritme, svaki korak ili aktivnost ima svrhu kojoj treba služiti, a oni bi trebali imati ispravan red kako bi algoritam radio onako kako mi želimo. Gornji opis se također naziva *pseudokôd*, koji se koristi za objašnjenje procesa praćenog algoritmom.

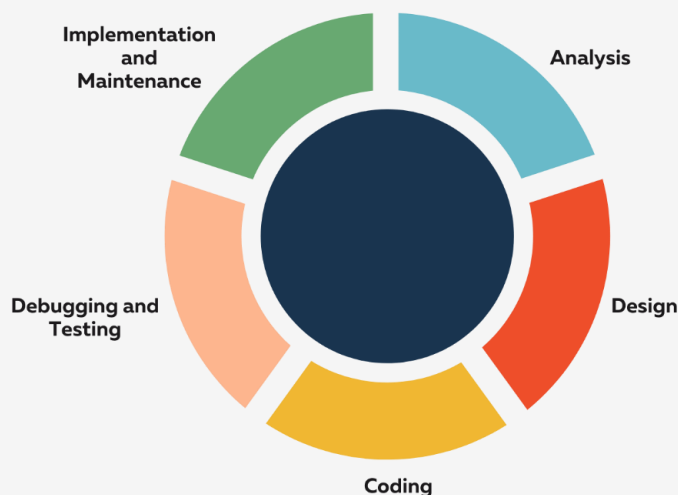
Drugi važan aspekt je obrazac koji se slijedi za dizajniranje specifičnog algoritma, koji obično uključuje obrazac ulaz, proces i izlaz (IPO). Algoritam počinje čitanjem podataka, prelazi na manipulaciju podacima i završava prikazivanjem rezultata. Na primjer, pretpostavimo da želite izračunati ocjene svojih učenika: prvo biste pregledali njihove rezultate na svakom testu (ulaz), zatim izračunali njihovu prosječnu ocjenu (proces) i na kraju pokazali njihovu prosječnu ocjenu (izlaz). Ovo je vrlo koristan obrazac koji treba uzeti u obzir pri izradi algoritma.



Slika 2 – IPO obrazac

Budući da smo naučili što je algoritam i proces koji je potreban za stvaranje dobrog algoritma, sada možemo malo bolje razumjeti kako razviti program. Razvoj programa prolazi kroz ciklus

koji uključuje analizu, dizajn, kodiranje, ispravljanje pogrešaka i testiranje, te implementaciju i održavanje predloženog rješenja.



**Slika 3** – Ciklus razvoja programa

U prvom koraku pokušavamo razumjeti, identificirati i analizirati problem koji zahtijeva računsko rješenje. Drugi korak se odnosi na dizajniranje programa, što podrazumijeva izradu vizualnog dijagrama procesa kojeg će program slijediti. Ovo je osobito korisno kako bi vam pomoglo razbiti problem na manje dijelove. Sljedeći korak je kodiranje programa na temelju vizualnog dijagrama kojeg smo izradili u prethodnom koraku. Korisnik računala će pokrenuti kôd za ovaj korak kako bi uočio bilo kakav problem. U četvrtom koraku korisnik mora započeti s ispravljanjem pogrešaka u programu kako bi se izbjegle moguće pogreške. Peti korak zahtijeva od korisnika da pokrene program i uvjeri se da ne postoje sintaktičke ili logičke pogreške. Posljednji i šesti korak vrti se oko dokumentacije i održavanja programa, što zahtijeva objašnjenje logike programa i njegovih procesa.

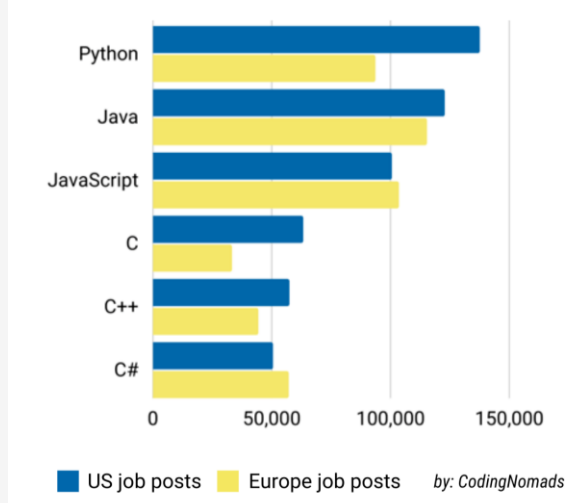
Ako postoje neke situacije koje niste razumjeli iz ovog procesa, ne brinite. Kad počnemo vježbati, postat će vam jasnije. U ovom modulu ćemo se usredotočiti na prva tri koraka. Stoga ćemo pokušati objasniti proces analize problema i izrade programa koji može poslužiti kao potencijalno rješenje.

#### 1.4. Programski jezici – Najtraženiji programski jezici

Prema analizi tisuća otvorenih radnih mjesta u SAD-u i Europi, Python je rangiran kao najtraženiji jezik za kodiranje za 2022. godinu, a slijede ga Java i JavaScript. Iako su rezultati sugerirali da potražnja za C, C++ i C# nije tako jaka, još uvijek je postojala za odabrana zanimanja. Iako Python postoji već desetljećima, njegova potražnja u 2022. nastavit će rasti zahvaljujući eksponencijalnoj upotrebi u uspješnim industrijama koje se bave znanosti o podacima, strojnom učenju i AI.



## Most in-demand programming languages 2021-2022



Slika 4 – Najtraženiji programski jezici

Izvor: <https://www.siliconrepublic.com/careers/python-most-in-demand-coding-language-2022>

## 2. Postavljanje Pytona

- ⇒ **Broj sudionika: 1-10 po voditelju**
- ⇒ **Trajanje: 0.75-1 sati**
- ⇒ **Metode poučavanja:** prezentacija, vođene instrukcije, iskustveno učenje
- ⇒ **Potrebni materijali:** prezentacija, računala (1 po sudioniku) I stabilna internetska veza

Prije nego što objasnimo kako postaviti integrirano razvojno okruženje (IDE) Python i kako pokrenuti Python u naredbenom retku, ukratko ćemo objasniti što je Python i zašto je najkorišteniji programski jezik na tržištu rada.

### 2.1. Što je Python?

Python je programski jezik visoke razine koji omogućuje veću fleksibilnost i čitljivost prilikom kodiranja i može se lako prilagoditi različitim vrstama računala bez ikakvih ili s vrlo malo mogućih izmjena. Ostali jezici visoke razine uključuju C, C++ i Java.

Razlika između programskih jezika visoke razine i programskih jezika niske razine je u tome što računala izvršavaju programe napisane na jezicima niske razine. Jezici niske razine koriste binarno kodiranje (0,1) za izvršavanje programa. Međutim, ljudima to nije lako protumačiti osim ako u tome nisu stručni. Stoga se programi koji su napisani na jezicima visoke razine moraju transformirati ili prevesti u binarni oblik kako bi računalo razumjelo što treba učiniti. To zahtijeva dodatno vrijeme obrade, ali to je relativno mali nedostatak jezika visoke razine.

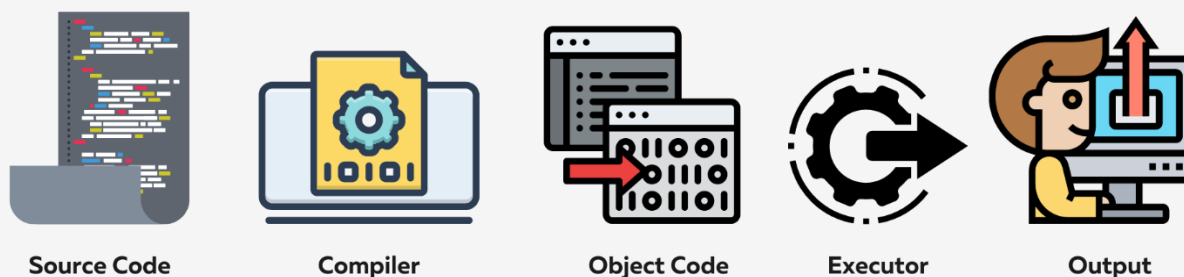
Općenito, postoje dva načina na koja se program prevodi u binarni oblik kako bi se izvršio. Prvi uključuje korištenje prevodilaca gdje se program čita i prevodi prije nego što se pokrene.



Kao što je prikazano na donjoj slici, proces počinje od izvornog kôda (za program visoke razine) prelazi na objektni kôd ili izvršitelj (koji prevodi program u binarni oblik) i vraća se natrag na sastavljanje programa koji će se izvršiti kao ishod bez daljnjeg prijevoda.

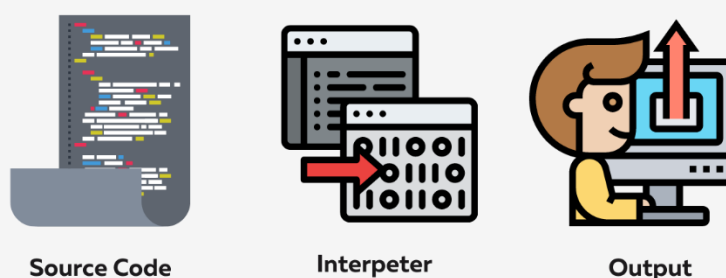
**Slika 5** - Jezik prevoditelja (preuzeto sa Downey et al., 2002, str.30)

Drugi način koristi samo tumača kao posrednika između izvornog kôda i izlaza. Tumač izvorno



čita program i slijedi njegove doslovne upute.

Ovo je proces naprijed-natrag koji traje dok se program ne završi, kao što možete vidjeti na donjoj slici.



**Slika 6** - Jezik tumača (preuzeto iz Downey et al., 2002., str. 30)

Python koristi tumača za izvršavanje svojih programa i stoga se smatra interpretiranim jezikom. To se može učiniti načinom naredbenog retka ili načinom rada skripte. U načinu naredbenog retka vi pišete svoj kôd, a tumač ispisuje krajnji rezultat. S druge strane, način rada skripte koristi program koji sadrži datoteku koja završava na .py i koristi tumač za izvršavanje sadržaja datoteke. Treba napomenuti da većina Python programa završava na .py, međutim, to ovisi o integriranom razvojnom okruženju (IDE) koje koristite.

U ovom modulu svi primjeri koriste način naredbenog retka za trenutno izvršavanje programa i bolje razumijevanje procesa. Nakon što dovršite program, možete ga spremiti kako biste ga kasnije mogli pokrenuti kao skriptu.

## 2.2. Najčešće korištena integrirana razvojna okruženja (IDEs) po industriji

U prethodnom odjeljku spomenuli smo integrirana razvojna okruženja (IDE), dopustite nam da ukratko objasnimo što je IDE. IDE je u osnovi softverska aplikacija koja sadrži različite mogućnosti računalnih programera za razvoj softvera kao što je uređivač izvornog kôda, alati za automatizaciju izgradnje i program za ispravljanje pogrešaka. Ako vam se čini da sve to

trenutno nema previše smisla, ne brinite. Kada počnemo vježbati, bolje ćete razumjeti sve što smo do sada objasnili.

Ovisno o industriji, preferiraju se različiti Python IDE-ovi. Kada tvrtka ili organizacija biraju IDE uzimaju se u obzir neki od čimbenika: namjera upotrebe IDE-a kao što su razvoj web/a, znanost o podacima, skriptiranje ili osiguranje kvalitete; operativni sustav koji koristi (npr. Linux/macOS, Windows ili mješoviti OS); je li hardver dobar ili loš; i razinu osobe koja će kodirati (npr. početnik, srednji ili napredni).



**Slika 7** - Najčešće korišteni IDE-ovi po industriji

(Preuzeto sa <https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/>)

### 2.3. Postavljanje Python integriranog razvojnog okruženja (IDE)

U ovom modulu možete koristiti Spyder (<https://www.anaconda.com/products/individual>) ili Visual Studio Code (<https://code.visualstudio.com/>) jer njih obično koriste početnici kako bi se upoznali s Pythonom u korisničkom okruženju.

Međutim, ako se niste spremni posvetiti IDE-u, također imate mogućnost korištenja mrežnih uređivača teksta kao što su sljedeći:

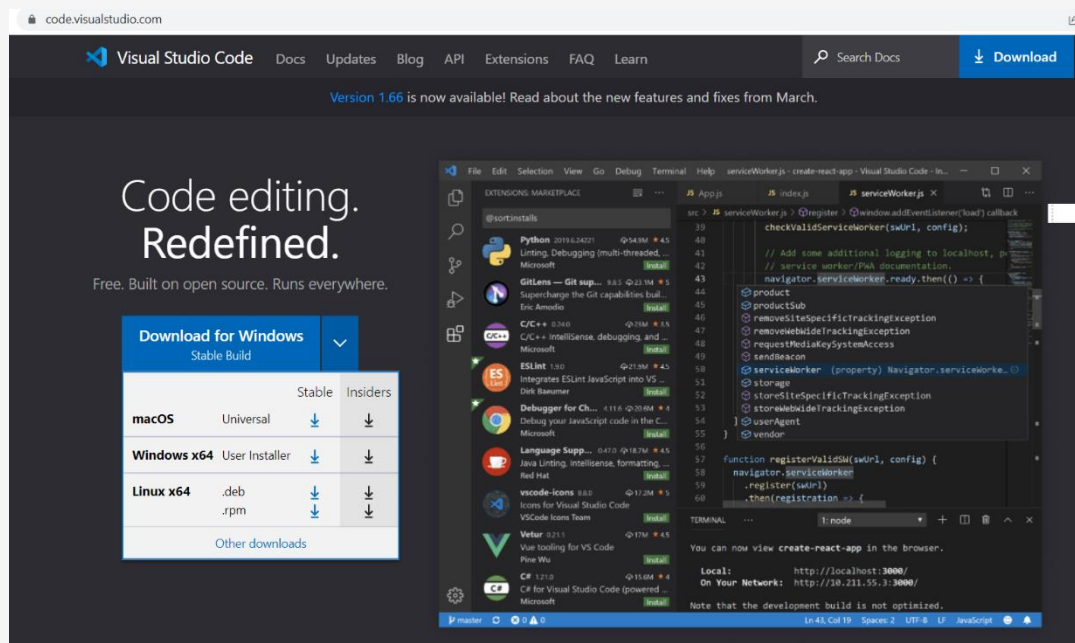
- [Python.org](https://python.org)
- [Programiz](https://programiz.com)
- [Tutorialspoint](https://tutorialspoint.com)

Preporučujemo korištenje IDE-a budući da je lakše pratiti svoje datoteke i napredak kada počnete kodirati.

#### 2.3.1 Instalacija Visual Studio Code

1. Idite na ovu web stranicu: <https://code.visualstudio.com/>

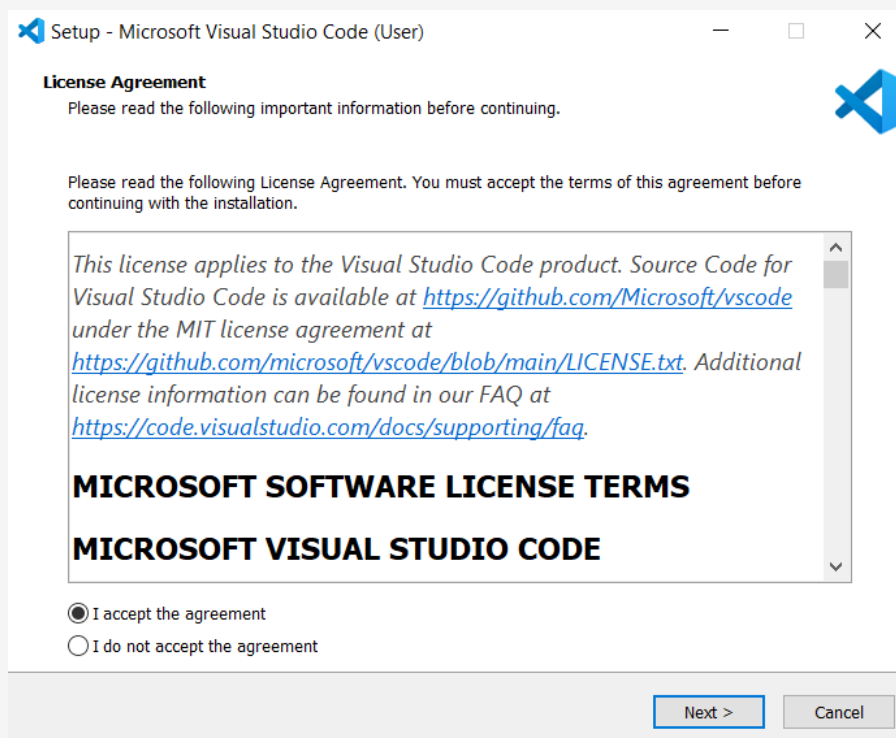
2. Nakon što se stranica učita, vidjet ćete gumb Preuzmi. Nakon što se stranica učita, vidjet ćete gumb Preuzmi. Automatski odabire vaš operativni sustav; međutim, možete kliknuti na strelicu pored nje i promijeniti je prema svojim potrebama.



Obavezno preuzmite stabilnu izradu!

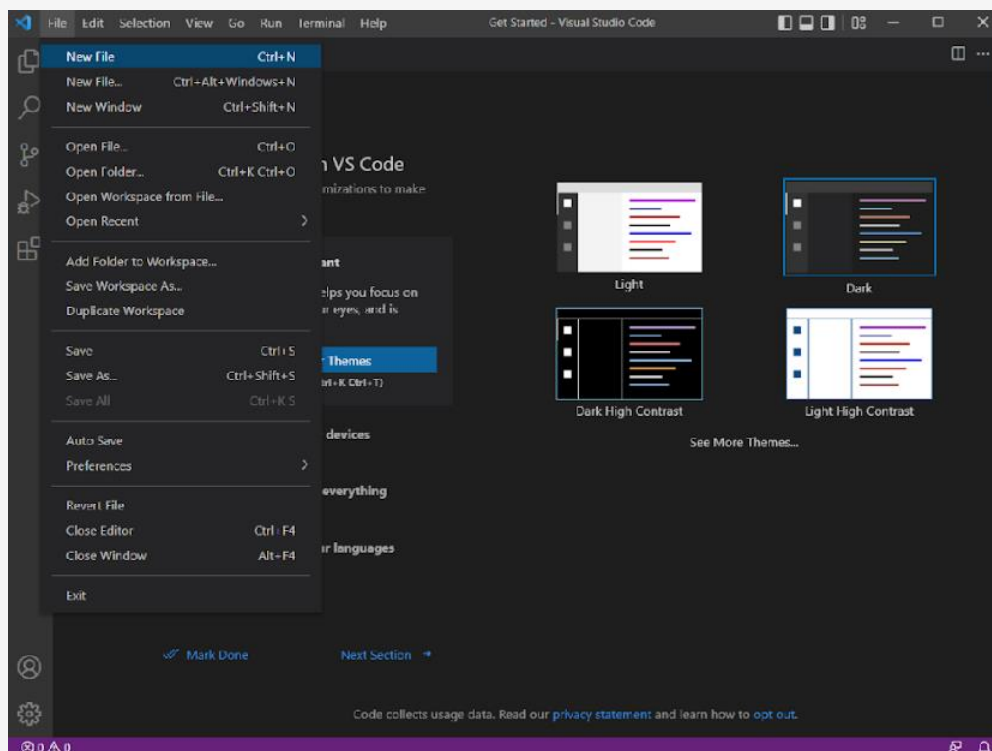
Slika 8 – Preuzimanje Visual Studio Coda

3. Kada je preuzimanje dovršeno, otvorite datoteku i slijedite upute za postavljanje.



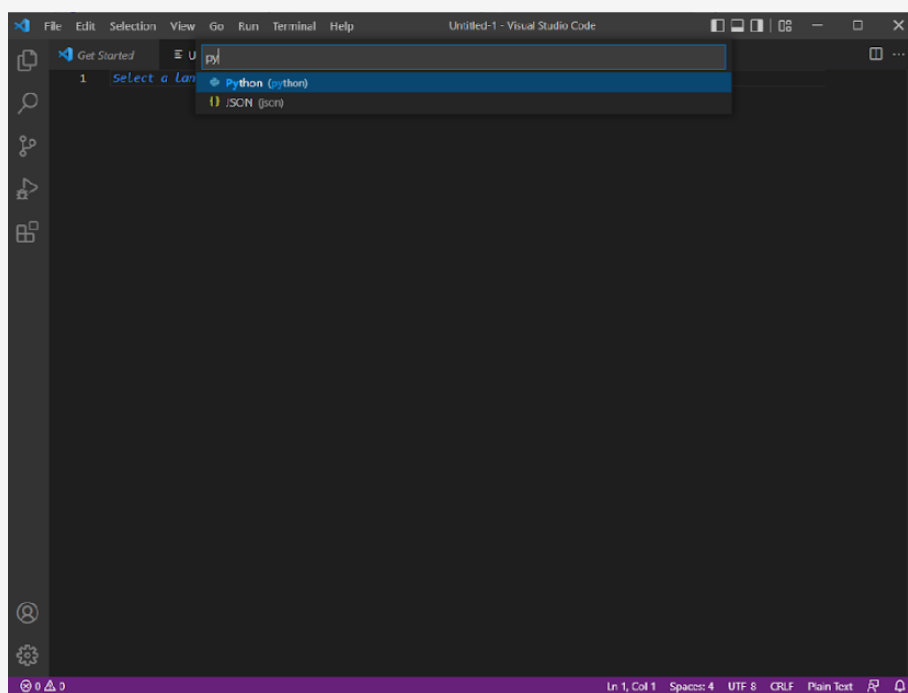
Slika 9 – Instalacija IDE

4. Nakon što program završi instalaciju, možete ga otvoriti ili će se automatski pokrenuti.
5. Odaberite Datoteka ▢ Novo



Slika 10 – Izrada nove datoteke u Visual Studio Co

6. Kliknite na Odaberite jezik i odaberite Python



Slika 11 – Odabir programskog jezika

## 7. Nakon što odaberete Python, bit ćete preusmjereni na instalaciju Pythona



Slika 12 – Instaliranje Python ekstenzija

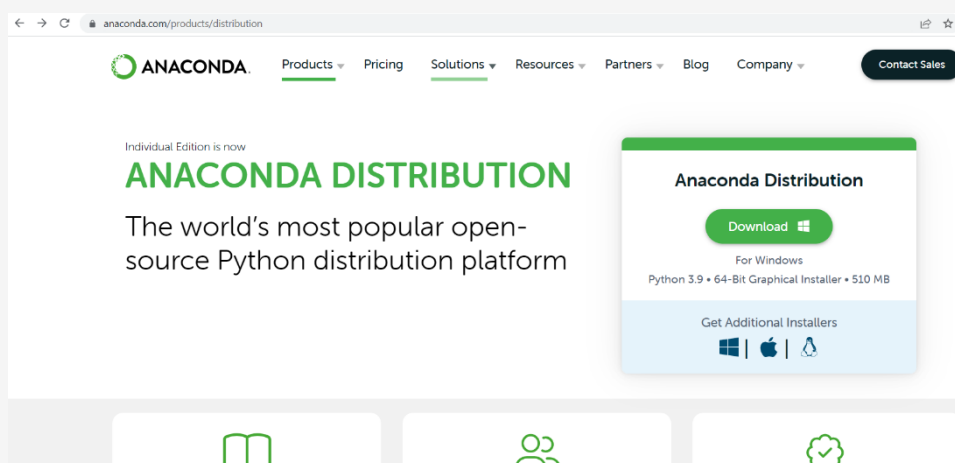
## 8. Nakon što instalirate Python, spremni ste za početak kodiranja!

Za više informacija, također možete konzultirati njihovu web stranicu o tome kako započeti:

<https://code.visualstudio.com/docs/introvideos/basics>

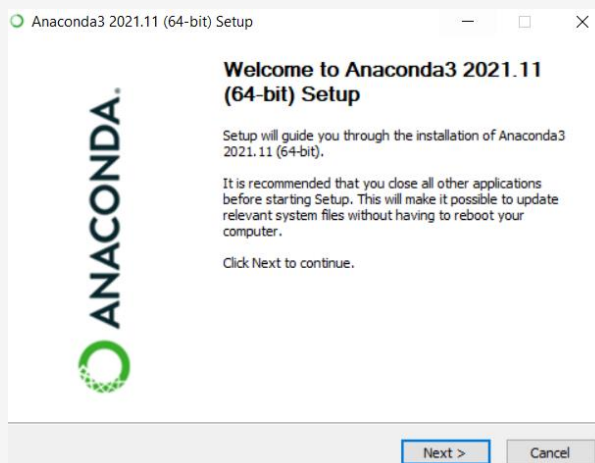
### 2.3.2. Instaliranje Spyder-a iz Anaconda-e

1. Idite na ovu web stranicu: <https://www.anaconda.com/products/individual>
2. Nakon što se stranica učita, vidjet ćete gumb Preuzmi. Ovisno o vašem operativnom sustavu, odaberite odgovarajuću verziju. Preporučujemo 64-bitni grafički instalacijski program.



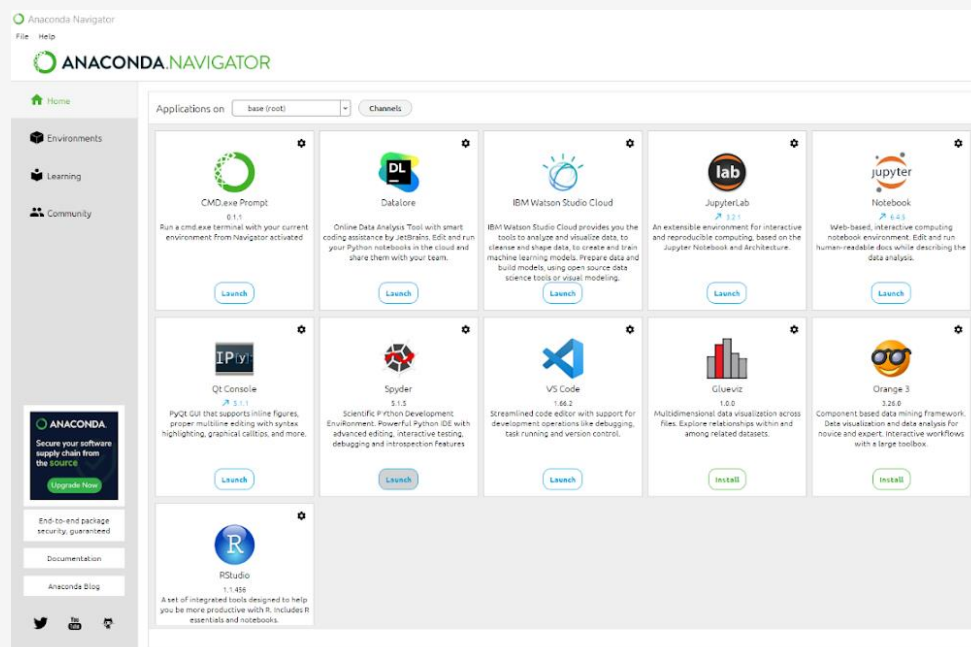
Slika 13 – Preuzimanje Anaconda Distribution

- a. Za više informacija o koracima potrebnim za Windows, slijedite ovu poveznicu: <https://docs.anaconda.com/anaconda/install/windows/>
  - b. Za korisnike macOS-a, slijedite ovu poveznicu: <https://docs.anaconda.com/anaconda/install/mac-os/>
  - c. Za korisnike Linuxa slijedite ovu poveznicu: <https://docs.anaconda.com/anaconda/install/linux/>
3. Kada je preuzimanje dovršeno, otvorite datoteku i slijedite upute za postavljanje.



Slika 14 – Instaliranje Anaconda-e

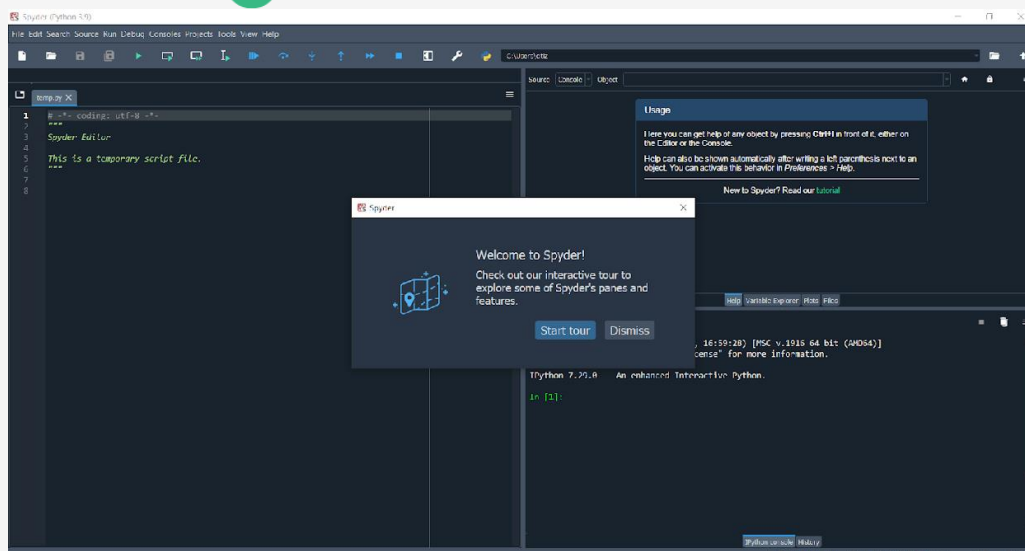
4. Nakon što program završi instalaciju, možete otvoriti Anaconda Navigator.
5. Nakon što se Navigator učita, odaberite aplikaciju Spyder i kliknite Pokreni. Ako nije



instaliran, kliknite gumb Instaliraj da biste instalirali Spyder.

Slika 15 – Anaconda Navigator

6. Kada otvorite Spyder, možete odabrati opciju da krenete u obilazak ili odbacite.



Slika 16 – Otvaranje Spyder-a po prvi put

## 7. Sada možete započeti s kodiranjem!

Za više informacija, također možete koristiti Anaconda početnicu: <https://docs.anaconda.com/anaconda/user-guide/getting-started/>

Sigurni smo da ste uspjeli lako dovršiti ovaj korak!

### 2.4. Pokretanje Pythona u naredbenom retku

Sada je vrijeme da pokrenemo naš prvi program. Možda se pitate je li to tako jednostavno. Odgovor je: Da, jeste!

Jednostavno upišite sljedeće:

```
print("Hello, World!")
```

Sada ga možete pokrenuti. Trebalo bi se ispisati na konzoli: **Hello, World!**

### 3. Osnove programiranja u Pythonu

- ⇒ **Broj sudionika:** 1-10 po voditelju
- ⇒ **Trajanje:** 8.00 sati
- ⇒ **Metode podučavanja:** prezentacija, vođene instrukcije, iskustveno učenje
- ⇒ **Potrebni materijali:** prezentacija, računala (1 po sudioniku) I stabilna internetska veza

Nakon što ste uspješno pokrenuli svoj prvi program, nastavit ćemo učiti više o osnovnim funkcijama Pythona i logici iza njih.

#### 3.1. Osnove: Vrste podataka (osnovne i složene)

Tipovi podataka važan su koncept u svijetu programiranja budući da oni određuju kako se podacima može manipulirati. Ugrađeni tipovi podataka u Pythonu spadaju u sljedeće kategorije:

General groups of data types	Specific data types used in Python	Examples
<b>Text</b>	str	"Hello, World!" (serije žica, uvijek upotrebljavajte navodnike)
<b>Numeric</b>	int, float, complex	3 (integer-int), 3.2 (decimalni brojevi-uključuje decimale), 2j+3 (uključuje i znakove i brojeve)
<b>Sequence</b>	list, tuple, range	["apple", "cherry", 1, 1] (popis; sekvence elemenata bilo koje vrste), () (tuple; određeni skup), raspon(5) (ponavlja niz brojeva, ima početnu i završnu točku (5))
<b>Mapping</b>	dict	{"brand": "Ford"} (rječnik; neporedana zbirka parova ključ-vrijednost)
<b>Set</b>	set, frozenset	{"apple", "cherry", 1} (skup; neporedana zbirka ne poduplanih stavki bilo koje vrste, odvojena zarezima)
<b>Boolean</b>	bool	True or False
<b>Binary</b>	bytes, bytearray, memoryview	Uobičajeno upravlja binarnim podacima

**Tablica 2** – Različite vrste podataka Pythona s primjerima (preuzeto sa [https://www.w3schools.com/python/python\\_datatypes.asp](https://www.w3schools.com/python/python_datatypes.asp))



Ovo su svi ugrađeni tipovi podataka koji postoje u Pythonu. Proći ćemo kroz sve ove osim zadnje grupe, binarne, budući da su ostale najkorisnije za razumijevanje početnicima.

### 3.2. Varijable, izrazi i iskazi

#### Varijable

*Varijable* u osnovi sadrže podatke koje im dodjeljujemo u obliku imena kako bismo naš kôd učinili čitljivijim i mogli koristiti određenu varijablu više od jednog puta.

Upisujemo ime koje želimo dodijeliti našoj varijabli, koristimo znak jednakosti (=) i stavljamo vrijednost koju želimo pohraniti na desnu stranu znaka jednakosti.

Primjer:

```
x = 5
print(x)
```

Ovdje je varijabla x jednaka 5 i print(x) ispisuje vrijednost sadržanu u x (tj. 5).

**\*Imajte na umu da je Python osjetljiv na velika i mala slova, stoga se x i X neće smatrati istom varijablom.**

Upamtite tipove podataka koje smo upravo vidjeli, možete napisati sljedeće da vidite tip podataka varijable x koju smo upravo kreirali:

```
print(type(x))
```

Konzola će prikazati: **int**

Pretpostavimo da želite stvoriti 3 varijable gdje jedna sadrži cijeli broj, druga decimalni i još jedna niz. Napisali bi sljedeće:

```
age = 35
name = "John"
price = 12.99
```

Ako sada pokušamo ovo pokrenuti, konzola neće pokazati ništa. Možda se pitate zašto je to tako, jednostavno zato što nismo pozvali varijablu, već smo joj samo dodijelili vrijednost. Kako bismo vidjeli što naša varijabla sadrži, moramo upotrijebiti **print (ime varijable)** kako bismo uputili Python da je prikaže. Inače neće.

Primjer:

```
print(age)
print(name)
print(price)
```

Ili ih možete ispisati sve zajedno na sljedeći način:

```
print(age, name, price)
```



Svojoj varijabli možemo dati bilo koje ime i nismo ograničeni samo na jednu riječ.

Međutim, ne biste trebali koristiti predugačka imena varijabli jer to nije prikladno ni vama ni čitatelju.

Nastojte da nazivi varijabli budu kratki i razumljivi. Ako želite da vaše ime varijable budu dvije odvojene riječi, jednostavno upotrijebite donju crtu između, tj. `my_age`.

**\*Imajte na umu da Python ne dopušta da nazivi varijabli počinju brojevima ili da uključuju posebne znakove kao što su @, #, \$, itd.**

Postoje i neke druge ključne riječi koje se koriste u Pythonu koje se ne mogu koristiti kao nazivi varijabli, jer su rezervirane za određene funkcije. Ove ključne riječi su:

<code>and</code>	<code>def</code>	<code>exec</code>	<code>if</code>	<code>not</code>	<code>return</code>
<code>assert</code>	<code>del</code>	<code>finally</code>	<code>import</code>	<code>or</code>	<code>try</code>
<code>break</code>	<code>elif</code>	<code>for</code>	<code>in</code>	<code>pass</code>	<code>while</code>
<code>class</code>	<code>else</code>	<code>from</code>	<code>is</code>	<code>print</code>	<code>yield</code>
<code>continue</code>	<code>except</code>	<code>global</code>	<code>lambda</code>	<code>raise</code>	

Tablica 3 – Rezervirane ključne riječi u Pythonu (preuzeto sa Downey et al., 2002., str. 15)

## Tvrđnje

Vidjeli smo dvije tvrdnje korištene u Pythonu do sada: `print` i dodjela varijabli. Naredbe koje se daju tumaču Pythona kako bi se izvršile, nazivaju se tvrdnjama.

Kako smo već spomenuli, tvrdnje, kao što je dodjela varijabli, ne daju rezultat. Međutim, ispisana naredba daje rezultat, a to je vrijednost varijable.

Niz tvrdnji smatra se skriptom koja ili daje rezultate ili ne, ovisno o vrsti korištenih tvrdnji. Rezultati se prikazuju na temelju redoslijeda tvrdnji.

Na primjer, ako upišete sljedeće tvrdnje:

```
print(name)
height_cm = 1.75
print(age)
```

Dobili biste kao ishod:

John

35

Varijabla visine sprema se samo kao varijabla, ali ne daje nikakav rezultat.

## Izrazi

U Pythonu se koriste različite vrste izraza. Izrazi uključuju kombinaciju različitih varijabli, operatora i vrijednosti (operandi) koji proizvode drugu vrijednost kao ishod.

Kao primjer, pokušajmo dodati dva broja bez dodjeljivanja imena varijabli ili upotrijebimo naredbu za ispis:

```
2+2
```

Ishod:

Ovdje možemo vidjeti da Python neće proizvesti ishod. Stoga je potrebno imati varijable, operatore i vrijednosti da bismo imali izraz.

Ako želite imati rezultat od 2+2, tada trebate ili uključiti naredbu za ispis ili dodijeliti naziv varijable i ispisati je.

Primjer:

```
print(2+2)
addition = 2+2
print(addition)
```

U primjeru 2+2, tražimo od Pythona da procijeni izraz. Ali kada bismo jednostavno napisali 2 i 2 u odvojene retke bez operatora (+), ne bi se proizveo nikakav rezultat. Čak i ako se izjava smatra legalnom, ona ne daje ishod.

Razmotrite sljedeći zapis:

```
56
4.5
"John"
print(3+2)
```

Ovdje biste jedino dobili rezultat 5 iz posljednjeg retka od 3+2. Što možemo dodati zapisu da prikažemo sve izraze na konzoli kao ishod?

*Savjet: do sada smo ga već koristili više puta*

## Operateri

Vrijeme je da objasnimo neke od operatora koji se koriste u Pythonu, kao što je znak zbrajanja (+) koji smo koristili ranije. Operateri su posebni simboli koji predstavljaju matematička izračunavanja, a to su:

addition (3+20)    subtraction (x-6)    multiplication (3\*5)    division (10/5)    exponentiation (4\*\*2)



Također možete koristiti nazive varijabli koji sadrže cijele brojeve ili decimalne brojeve umjesto običnih cijelih brojeva za izvođenje operacija.

Primjer:

```
course_grade1 = 15
course_grade2 = 18
coursesum = course_grade1+course_grade2
print(coursesum)
```

Ishod: 33

Kako bismo pronašli prosječnu ocjenu na temelju ocjene 2 kolegija, moramo ih zbrojiti i zatim podijeliti s 2. Što mislite, na koji se način to može učiniti? Odvojite trenutak da razmislite o tome.

Odgovor:

```
courseavg = (course_grade1+course_grade2)/2
print(courseavg)
```

Ishod: 16.5

Ovaj primjer također pokazuje da Python prepoznaje redoslijed operacija, što nalaže da zagrada ima najveći prioritet umjesto podjele koja se nalazi izvan zagrada.

U Pythonu prednost je kako slijedi: 1) zagrada, 2) eksponencijacija, 3) množenje/dijeljenje i 4) zbrajanje/oduzimanje.

Treba napomenuti da kada operatori imaju isti prioritet, oni se ocjenjuju s lijeva na desno. Na primjer, razmotrite sljedeću operaciju:  $5*30/60$ . To je jednako 2.5, što pokazuje da se prvo događa množenje (=150), a zatim dijeljenje koje daje krajnji rezultat.

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_variables1](https://www.w3schools.com/python/exercise.asp?filename=exercise_variables1)

## Operateri koji djeluju na nizovima

Čak i ako se niz smatra brojem (tj. '15'), općenito ne možete izvoditi gore spomenute operacije na nizovima.

Međutim, operateri zbrajanja (+) i množenja (\*) imaju drugačiji učinak kada se primjenjuju na nizove. Ako koristite zbrajanje između dvije varijable koje sadrže nizove, one će biti povezane. To znači da će biti spojeni kao jedan slijed nizova.

Primjer:

```
first_name = 'Emma'
last_name = 'Smith'
print(first_name + last_name)
```



Ishod: Emma Smith

\* Imajte na umu da ako navodnici i nizovi imaju bilo kakav razmak između sebe, na kraju ćete dobiti "EmmaSmith" umjesto "Emma Smith". Stoga su razmaci također dio nizova.

Kada se koristi na nizovima, operater množenja (\*) izvodi ponavljanje.

Primjer:

```
print("First"*3)
```

Ishod: FirstFirstFirst

Da bi ovo funkcioniralo, trebate upotrijebiti cijeli broj da odredite koliko puta treba ponoviti niz.

### Komentari

Kako programi rastu, postaje sve teže pronaći i razumjeti algoritme koji se koriste za dobivanje krajnjeg rezultata. Stoga su komentari korisni za objašnjenje logike i procesa programa na prirodnom jeziku. Razmislite o primjedbama poput dodavanja bilješki gdje god je potrebno kako bi vaš kôd bio čitljiviji, samo morate dodati simbol ljestve (#) za male komentare.

Primjer:

```
#calculating the average grade of students based on the course grade
courseavg = (course_grade1+course_grade2)/2
```

Također možete dodati komentar u isti redak kao kôd:

```
print(first_name + last_name) #concatenating two strings
```

Što god napišete nakon simbola ljestve (#), zanemaruje se i program ga ne izvršava kao kôd.

Druga opcija koja se može koristiti za veće dijelove teksta je dodavanje 3 navodnika (""") prije i 3 (""") nakon što je tekst gotov.

Primjer:

```
"""
I can add as much text as I like to explain a function
"""
```

Korištenje komentara korisna je praksa za svakog stručnjaka koji radi s kodiranjem.

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_comments1](https://www.w3schools.com/python/exercise.asp?filename=exercise_comments1)

### 3.3. Popisi, rječnici i tuplei

U ovom ćemo odjeljku vidjeti kako koristiti popise, rječnike i tuple u Pythonu. Kao što smo ranije vidjeli u odjeljku o vrstama podataka, popisi i tuplei spadaju u grupu sekvenci, a rječnici



unutar grupe mapiranja. Sve ove vrste podataka imaju različite značajke i kvalitete koje omogućuju manipulaciju podacima na različite načine

### 3.3.1. Popisi

Popisi se pokreću korištenjem uglastih zagrada [] i sadrže niz poredanih stavki u jednoj varijabli.

Primjer:

```
adj = ["smart", "beautiful", "stunning"]
```

Stavke uključene u popis su uređene, sklone miješanju (tj. promjenjive) i dopuštaju duplicirane vrijednosti.

Svaka stavka na popisu identificirana je indeksom. Indeks počinje od [0] i svaki put se povećava za jedan; dakle, prva stavka na popisu ima indeks [0], a druga [1] i tako dalje. Treba napomenuti da se njihov redoslijed ne može mijenjati. Kada se dodaju nove stavke, one će biti stavljene na kraj popisa.

Također, popisi su promjenjivi, što znači da se stavke mogu mijenjati, uklanjati ili dodavati nakon što je popis stvoren. Budući da dopuštaju duplicirane vrijednosti, možete pronaći vrijednost više puta na popisu.

Primjer:

```
fruits = ["apple", "banana", "apple", "grapefruit"]
```

Popisi također mogu sadržavati kombinaciju cijelih brojeva, nizova ili logičkih vrijednosti:

```
employee1 = ["John Smith", 35, "male", 25000, True]
```

Ako želite odrediti koliko je brojeva na popisu, možete upotrijebiti funkciju `len()`:

```
print(len(employee1))
```

**\* Imajte na umu da se broj zagrada mora podudarati od početka do kraja kako bi se kôd mogao izvršiti.**

Primijetite da u ovom primjeru, imamo jedan kôd koji počinje na funkciji `len()` i jedan unutar popisa, te zbog toga vidite dvije završne zagrade na kraju kôda.

Također možete koristiti funkciju `list()` za pokretanje popisa:

```
new_list = list(("tomato", "cucumber", "peas", "peppers"))
print(new_list)
```

**\* Primijetite da ovdje trebate dodati dvostruku zagradu kada koristite funkciju `list()` za stvaranje popisa.**



## Pristupanje stavkama na popisima

Ako želite pristupiti stavci na popisu, trebate koristiti njihov indeksni broj.

Upotrijebimo popis `employee1` (zaposlenik1) kojeg smo ranije kreirali da dobijemo dob zaposlenika, što je druga stavka na popisu, u sljedećem primjeru:

```
print(employee1[1])
```

**\* Sjećate li se zašto koristimo 1 umjesto 2 kao indeksni broj za drugu stavku na popisu? To je zato što indeks počinje od 0 na popisima.**

Ako imate dug popis stavki i želite dobiti posljednju stavku s popisa, možete koristiti negativno indeksiranje. To znači da će koristiti `-1` za pristup posljednjoj pronađenoj stavci na popisu, `-2` da dobije drugu posljednju stavku i tako dalje.

Primjer:

```
print(employee1[-1])
```

Također možete odrediti raspon indeksa koji označava od kojeg broja indeksa započeti i do kojeg broja indeksa završiti raspon.

Primjer:

```
print(employee1[2:4])
```

U ovom primjeru će početi od treće stavke na popisu i završiti na posljednjoj jer na popisu imamo samo 5 stavki. Uvijek zapamtite da numeriranje počinje od 0.

Ako odlučite da ne želite odrediti početni broj indeksa, raspon će početi od prve pronađene stavke na popisu:

```
print(employee1[:4])
```

Također možete odrediti samo početni broj indeksa bez završnog indeksnog broja:

```
print(employee1[1:])
```

Sjetite se kada smo spominjali negativno indeksiranje, možete ga koristiti i za pristup stavkama na popisu s kraja popisa:

```
print(employee1[-4:-1])
```

Time ćemo doći do četvrte stavke od kraja do posljednje pronađene na popisu.

Također možete provjeriti postoji li stavka na popisu. S popisa pridjeva koji smo ranije izradili, recimo da želimo provjeriti je li stavka "beautiful" (lijepo) uključena u popis:



```
adj = ["smart", "beautiful", "stunning"]
if "beautiful" in adj:
    print("Yes, beautiful is included in the list")
```

## Dodavanje stavki na popise

Ako želite dodati nove stavke na kraj popisa, možete koristiti metodu **append()**. Recimo da želimo dodati novi pridjev na popis pridjeva, učinit ćemo sljedeće:

```
adj.append("innovative")
print(adj)
```

Također imate mogućnost dodavanja novih stavki navođenjem indeksnog broja. To možete učiniti korištenjem metode **insert()**:

```
adj.insert(1, "innovative")
print(adj)
```

Kao što možete vidjeti ovdje, prvo odredite broj indeksa, a zatim stavku koju želite dodati odvojenu zarezom.

Druga metoda koja se može koristiti pri dodavanju stavki s drugog popisa na trenutni popis je **extend()**. Recimo da imamo opći popis riječi i da želimo dodati pridjeve u opći popis riječi koji imamo:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.extend(adj)
print(words)
```

## Izmjena popisa

Ako želite izmijeniti određenu stavku koja se nalazi na popisu, trebete se pozvati na njen indeksni broj. Kao primjer, koristit ćemo popis `employee1` (zaposlenik1) kako bismo izmijenili dob zaposlenika:

```
employee1 = ["John Smith", 35, "male", 25000, True]
employee1[1] = 36
print(employee1)
```

Također možete promijeniti više stavki na popisu unutar određenog raspona. To možete učiniti definiranjem novih stavki koje će se dodati i uputiti na indeksne brojeve koje želite izmijeniti.

Na primjer, izmijenit ćemo prve tri stavke na popisu `employee1` (zaposlenik1), budući da taj zaposlenik više ne radi u toj tvrtki:





```
employee1[0:2] = ["Ella Smith", 30, "female"]
print(employee1)
```

### Uklanjanje stavki s popisa

Ako ste odlučili ukloniti navedenu stavku s popisa, možete koristiti metodu **remove()**. Recimo da želimo ukloniti booleovu vrijednost s popisa `employee1` (zaposlenik1):

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.remove(True)
print(employee1)
```

Također možete ukloniti stavku tako da navedete njen indeksni broj metodom **pop()**:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.pop(4)
print(employee1)
```

Ako ne navedete indeksni broj, posljednja stavka pronađena na popisu bit će uklonjena:

```
employee1.pop()
print(employee1)
```

Kao alternativu, možete koristiti ključnu riječ **del** za uklanjanje stavke s navedenim indeksom:

```
del employee1[1]
```

Također, možete koristiti **del** za brisanje cijelog popisa:

```
del employee1
```

Međutim, možda biste željeli isprazniti sadržaj popisa. U ovom slučaju možete koristiti metodu **clear()** me:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee1.clear()
print(employee1)
```

### Kopiranje popisa

Postoje dva načina na koje možete kopirati popis s novim imenom varijable. Prvi je korištenjem metode **copy()**:

```
employee1 = ["Ella Smith", 30, "female", 25000, True]
employee2 = employee1.copy()
print(employee2)
```

Drugi način na koji ovo može funkcionirati je korištenje metode **list()**:

```
employee2 = list(employee1)
print(employee2)
```

## Razvrstavanje popisa

Ako želite sortirati popis alfanumeričkim redoslijedom, bilo silazno ili uzlazno, možete koristiti metodu **sort()**.

\* Imajte na umu da će metoda **sort()** prikazati stavke prema zadanim postavkama, prema zadanom uzlaznom redoslijedu, ako nije drugačije navedeno.

Upotrijebimo popis riječi da ga sortiramo po abecednom redu u ovom primjeru:

```
adj = ["smart", "beautiful", "stunning"]
adj.sort()
print(adj)
```

Još jedan primjer koji uključuje numerički popis:

```
monthly_income= [5000, 1000, 2500, 1300, 1200, 1500, 2300]
monthly_income.sort()
print(monthly_income)
```

Ova dva primjera će biti prikazana uzlaznim redoslijedom, tj. od manjeg prema većem ili od a do z ovisno o tipovima podataka.

Ako želite sortirati popis u silaznom redoslijedu, morate ga navesti u zagradi pomoću ključne riječi **reverse = True**. Sjetite se kada smo spomenuli booleove kao tipove podataka, u ovom slučaju se koristi za omogućavanje obrnute opcije navodeći da je True (istinita), budući da je zadana vrijednost False (pogrešna).

Primjer 1:

```
adj = ["smart", "beautiful", "stunning"]
adj.sort(reverse=True)
print(adj)
```

Primjer 2:

```
monthly_income= [5000, 1000, 2500, 1300, 1200, 1500, 2300]
monthly_income.sort()
print(monthly_income)
```

Opcija Sort (sortiranje) je prema zadanim postavkama osjetljiva na velika i mala slova, što znači da su sva velika slova poredana prije malih slova.

U ovom primjeru možete dobiti neke iznenađujuće rezultate zbog ovoga:



```
words = ["I", "am", "good", "you", "are", "nice"]
words.sort()
print(words)
```

Da biste prevladali ovaj problem, možete koristiti funkciju sortiranja bez osjetljivosti na velika i mala slova, a to je `key = str.lower` (tipka = razv.manje). Ovdje se koristi niz tipa podataka kako bi se napravila mala slova.

Primjer:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.sort(key=str.lower)
print(words)
```

Osim toga, možete koristiti metodu `reverse()` kako biste rezervirali redoslijed popisa bez obzira na njegov abecedni red:

```
words = ["I", "am", "good", "you", "are", "nice"]
words.reverse()
print(words)
```

### Pridruživanje popisa

Sjetite se kada smo govorili o korištenju aritmetičkih operatere i njihov učinak na nizove. Jedna od metoda za spajanje popisa je korištenje **addition sign (+)** (oznaka zbrajanja (+)):

```
adj = ["smart", "beautiful", "stunning"]
words = ["I", "am", "good", "you", "are", "nice"]
all_words = adj + words
print(all_words)
```

Ovdje smo stvorili novi popis kombinirajući dva već postojeća popisa pridjeva i riječi. Bez obzira na tipove podataka koji se nalaze na bilo kojem od ova dva popisa, možete ih kombinirati s + operaterom.

Ako želite dodati stavke s jednog popisa na drugi, jednostavno upotrijebite metodu **extend()**:

```
words.extend(adj)
print(words)
```

U ovom primjeru, popis riječi spojen je s popisom pridjeva. Stoga će popis riječi sada sadržavati oba popisa.

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_lists1](https://www.w3schools.com/python/exercise.asp?filename=exercise_lists1)

### 3.3.2. Tuplei

Tuplei također sadrže niz stavki u jednoj varijabli poput popisa. Slično kao kod popisa, tuplei su također uređene. Međutim, jedna ključna razlika između popisa i torki je ta što su torke neizmjenjive (tj. nepromjenjive). Torke se nalaze u okruglim zagradama ().



Primjer:

```
coordinates = (38.09, -77.06)
```

U stvarnom svijetu, tuple se obično koriste kao rječnik bez ključeva za pohranu podataka budući da se brže ponavljaju i njihove stavke se ne mogu mijenjati. Torke također dopuštaju duplicirane vrijednosti.

Ako želite vidjeti koliko je stavki u tupleu, možete koristiti funkciju `len()`:

```
print(len(coordinates))
```

Kada se stvara tuple koji sadrži samo jednu stavku, trebate dodati zarez iza stavke. Inače, Python ga neće prepoznati kao tuple. Koristit ćemo funkciju "type" da odredimo razliku između korištenja zareza.

Primjer:

```
name1 = ("John",)
print(type(name1))

name2 = ("John") #not a tuple
print(type(name2))
```

Slično popisima, torke mogu sadržavati stavke bilo koje vrste podataka bilo u zasebnim torkama ili kao kombinaciju prikazanu u primjeru u nastavku:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1)
```

Alternativni način za stvaranje tuple je korištenje `tuple()` konstruktora:

```
coordinates = tuple((38.09, -77.06))
print(coordinates)
```

**\* Imajte na umu da kada koristite konstruktor za stvaranje tuple, morate dodati dvostruke okrugle zagrade.**

## Pristupanje tupleima

Stavke u torkama koriste indeksiranje, što znači da svaka stavka u torci odgovara broju. Indeksiranje se temelji na redoslijedu stavki i počinje od 0 za prvu stavku i svaki put se povećava za 1.

Da biste pristupili određenoj stavci, trebate koristiti uglate zagrade i navesti indeksni broj:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1)
```



\* Zapamtite da indeks broj 1 u torci odgovara drugoj stavci, tj. 34.

Također možete koristiti negativno indeksiranje kako bi pristupili posljednjoj stavci torke [-1] ili drugoj posljednjoj stavci [-2] i tako dalje.

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1[-1])
```

Ovo će prikazati "male" stavku u tuple.

Ako želite, možete odrediti raspon indeksa koji označava od kojeg indeksnog broja započeti i do kojeg broja indeksa završiti raspon.

Primjer:

```
personal_info1 = ("John Kent", 34, True, "male")
print(personal_info1[1:3])
```

U ovom će primjeru početi od treće stavke na popisu i završiti na posljednjoj, budući da imamo samo 5 stavki na popisu. Uvijek zapamtite da numeriranje počinje od 0.

Ako odlučite da ne želite navesti početni broj indeksa, raspon će započeti od prve stavke pronađene u tupleu:

```
print(personal_info1[:3])
```

Također možete specificirati početni broj indeksa bez krajnjeg broja indeksa:

```
print(personal_info1[1:])
```

Sjetite se kada smo uveli negativno indeksiranje, možete ga koristiti i za pristup stavkama na popisu s kraja popisa:

```
print(personal_info1[-3:-1])
```

Time ćemo dobiti četvrtu stavku od kraja do posljednje u tuple.

Možete također provjeriti postoji li stavka u tupleu, slično onome što smo vidjeli na popisima. Recimo da želimo provjeriti je li beautiful uključeno u adj (pridjev) tuple:

```
adj = ("smart", "beautiful", "stunning")
if "beautiful" in adj:
    print("Yes, beautiful is included in the adj tuple")
```

## Dodavanje, mijenjanje i uklanjanje stavki iz tuplea



Tehnički, ne možete dodavati ili uklanjati stavke iz torki budući da su one neizmjenjive (tj. nepromjenjive). Međutim, postoji rješenje. Samo trebate pretvoriti tuple u popis, izmijeniti ga i zatim ga pretvoriti natrag u torku.

Pogledajmo primjer kojim ćemo sve malo bolje razumjeti:

```
personal_info1 = ("John Kent", 34, True, "male")
#here we create a new variable that converts the tuple into a list
modify_pinfo1 = list(personal_info1)
modify_pinfo1[1] = 35 #modify/change the item that we want
personal_info1 = tuple(modify_pinfo1)
print(personal_info1)
```

Ako želite dodati stavke u tuple, postoje dva načina koja možete koristiti. Prvi slijedi istu logiku kao i primjer koji smo vidjeli za modificiranje stavke pronađene u tupleu.

Primjer:

```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = list(personal_info1) #convert tuple into list in a new variable
modify_pinfo1.append("1 Charming Street") #adding new item
personal_info1 = tuple(modify_pinfo1) #convert it back
```

Drugi način za dodavanje nove stavke je dodavanjem torke drugoj torci. Možete stvoriti novi tuple koji sadrži stavku koju želite dodati postojećem tupleu i jednostavno ju ubaciti:

```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = ("1 Charming Street",) #creating new tuple, do not forget the comma
personal_info1 += modify_pinfo1 #adding new tuple into the existing one
print(personal_info1)
```

Ovdje ćemo vidjeti kako možemo ukloniti stavke iz torki. Primjenjuje se ista logika kao u prethodnim primjerima dodavanja ili mijenjanja stavki u torkama.

Primjer:

```
personal_info1 = ("John Kent", 34, True, "male")
modify_pinfo1 = list(personal_info1) #convert tuple into list in a new variable
modify_pinfo1.remove("male") #delete item male
personal_info1 = tuple(modify_pinfo1) #convert back
print(personal_info1)
```

U sljedećem primjeru vidjet ćemo kako možete izbrisati tuple pomoću ključne riječi `del`:

```
personal_info1 = ("John Kent", 34, True, "male")
del personal_info1 #it is now deleted
print(personal_info1) # this will raise an error since we have deleted it
```

Prilikom stvaranja tuplea, dodjeljujemo vrijednosti ili stavke koje će on sadržavati. Ovaj proces se također naziva "pakiranje" tuple:

```
personal_info1 = ("John Kent", 34, True, "male")
```

U Pythonu također imate priliku izdvojiti vrijednosti u varijable, što se naziva raspakiranje:

```
personal_info1 = ("John Kent", 34, True, "male")
# assigned separate variable name to each tuple item
(name, age, employed, sex) = personal_info1
# printing each new item variable from tuple
print(name)
print(age)
print(employed)
print(sex)
```

\* Imajte na umu da nazivi varijabli moraju odgovarati stavkama sadržanim u tupleu, inače možete koristiti zvjezdicu (\*) kako biste prikupili ostatak kao popis.

Recimo da želite izdvojiti samo dvije varijable iz tuplea `personal_info1`, možete koristiti zvjezdicu:

```
personal_info1 = ("John Kent", 34, True, "male")
(name, *demographics) = personal_info1
# all items after the first will be contained in the demographics variable
print(name)
print(demographics)
```

Ako dodate zvjezdicu drugom nazivu varijable umjesto posljednjem, tada će vrijednosti biti dodijeljene varijabli sa zvjezdicom sve dok se preostale vrijednosti ne podudaraju s preostalim varijablama:

### Spajanje tuplea

Ako želite spojiti dvije torke, možete koristiti **operator zbrajanja (+)**:

```
personal_info1 = ("John Kent", 34, True, "male")
personal_info2 = ("Kylie Smith", 40, True, "female")
total_pinfo = personal_info1 + personal_info2
print(total_pinfo)
print(sex)
```

Također imate mogućnost množenja sadržaja tuplea nekoliko puta pomoću **operatora množenja (\*)**:

```
personal_info1 = ("John Kent", 34, True, "male")
general = personal_info1 * 2
print(general)
```

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_tuples1](https://www.w3schools.com/python/exercise.asp?filename=exercise_tuples1)

### 3.3.3. Rječnici

Rječnici spadaju u grupu za mapiranje tipova podataka budući da mogu pohraniti podatke u key:value pairs (tipki: spajanje vrijednosti). Oni sadrže niz parova elemenata koji su poredani,



izmjenjivi (tj. promjenjivi) i ne dopuštaju duplicirane vrijednosti. Rječnici su označeni vitičastim zagradama {}.

**\* Imajte na umu da rječnici u Pythonu 3.6 i ranijim verzijama nisu bili poredani, već su to postali od verzije Pythona 3.7.**

Primjer rječnika može biti za prevođenje engleskog na drugi jezik. Recimo da želimo prevesti s engleskog na španjolski:

```
eng2esp = {
    "hello": "hola",
    "good morning": "buenas dias",
    "good afternoon": "buenas tardes",
    "how are you": "como estas"
}
print(eng2esp)
```

Druga mogućnost je stvoriti prazan rječnik i dodavati elemente jedan po jedan:

```
eng2esp = {}
eng2esp ["hello"] = "hola"
eng2esp ["good morning"] = "buenas dias" # and so on
```

Kao što smo već spomenuli, rječnici ne dopuštaju duplicirane vrijednosti za svoje ključeve, tj. "hello", u eng2esp rječniku koji smo kreirali iznad.

Ako želite znati koliko se stavki nalazi u rječniku, možete koristiti funkciju **len()** kao što smo vidjeli za liste i torke:

```
print(len(eng2esp))
```

Još jednom, slično torkama i listama, rječnici mogu sadržavati bilo koju vrstu podataka:

```
sofas_inv = { "sofas": 5,
             "colours": [ "red", "blue", "grey" ],
             "year": [2016, 2022, 2018],
             "sofabed": False}
```

Kao što možete vidjeti ovdje, imamo tipove podataka int, strings, bool i list.

Da biste odredili tip podataka varijable sofas, koristite **type()**:

```
print(type(sofas_inv))
```

## Pristupanje stavkama u rječnicima

Ako želite pogledati što "hello" znači na španjolskom, onda možete koristiti sljedeće:

```
eng2esp = {
    "hello": "hola",
    "good morning": "buenas dias",
    "good afternoon": "buenas tardes",
    "how are you": "como estas"
}
print(eng2esp["hello"])
```



Ovo će ispisati: "hola"

Kada tražite stavke u rječniku, upotrijebite njihov ključni naziv unutar uglastih zagrada:

```
# here we created a variable to store the value "hola"
esp_hello = eng2esp["hello"]
```

Ovdje je "hello" ključ vrijednosti "hola".

Alternativna opcija je korištenje metode **get()** kako biste dobili ekvivalentni rezultat:

```
esp_hello = eng2esp.get("hello")
```

Ako želite vidjeti popis svih ključeva koje rječnik sadrži, možete koristiti metodu **keys()**:

```
all_keys = eng2esp.keys()
print(all_keys)
```

Sve promjene napravljene na rječniku eng2esp odrazit će se na popisu all\_keys koji smo kreirali:

```
eng2esp["bye"] = "adios"
print(all_keys) #after the addition
```

Također, imate mogućnost dobivanja svih vrijednosti sadržanih u rječniku uz pomoć metode **values()**:

```
all_values = eng2esp.values()
print(all_values)
```

Ista logika se odnosi za vrijednosti u smislu promjena. Ako dodamo novu vrijednost, ona će biti dodana na popis vrijednosti koje smo kreirali.

Druga opcija koju imate je dobivanje svih parova key:value (tipka:vrijednost) korištenjem metode **items()**:

```
all_pairs = eng2esp.items()
print(all_pairs)
```

Opet, sve promjene unesene u rječnik, bilo u njegovim ključevima ili vrijednostima, odrazit će se na popis all\_pairs kojeg smo kreirali.

Ako niste sigurni postoji li određeni ključ u rječniku, možete provjeriti:

```
if "hello" in eng2esp:
    print("Yes, 'hello' is one of the keys of the eng2esp dictionary")
```



## Dodavanje, zamjena ili uklanjanje stavki iz rječnika

Za dodavanje stavki u rječnik možete koristiti dva načina. Prvi koristi novi indeksni ključ i njegovu dodanu odgovarajuću vrijednost:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey" ],
              "year": [2016, 2022, 2018],
              "sofabled": False}
sofas_inv["material"] = "leather"
print(sofas_inv)
```

Drugi način je korištenje metode **update()**, gdje trebate navesti rječnik, ili iterativni par key:value (tipka:vrijednost):

```
sofas_inv.update({"material" : "leather"})
print(sofas_inv)
```

\* Imajte na umu da su vitičaste zagrade nakon zagrade potrebne za označavanje parova key:value (tipka:vrijednost) u rječniku.

Ako želite mijenjati stavke u rječniku, dostupne su iste opcije kao što je gore opisano. Prva opcija je upućivanje na naziv ključa za promjenu određene vrijednosti. Na primjer, ako je kauč prodan, možete promijeniti količinu sofe u trgovini:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey" ],
              "year": [2016, 2022, 2018],
              "sofabled": False}
sofas_inv["sofas"] = 4
```

Druga se opcija odnosi na metodu **update()**:

```
sofas_inv.update({"sofas": 4})
```

Postoje 3 različita načina za uklanjanje stavki iz rječnika. Prvi koristi metodu **pop()**:

```
sofas_inv = { "sofas": 5,
              "colours": [ "red", "blue", "grey" ],
              "year": [2016, 2022, 2018],
              "sofabled": False}
sofas_inv.pop("sofabled")
print(sofas_inv)
```

Drugi način je korištenje ključne riječi **del**:

```
del sofas_inv["sofabled"]
print(sofas_inv)
```

Treći dostupni način uklanja posljednju stavku dodanu u Pythonu 3.7. Međutim, u starijim verzijama umjesto toga će se ukloniti nasumična stavka.



Primjer:

```
sofas_inv.popitem()
print(sofas_inv)
```

Ako želite izbrisati rječnik u potpunosti, možete koristiti ključnu riječ **del**:

```
del sofas_inv
print(sofas_inv) #Python will raise error since the dictionary no longer exists
```

Druga opcija koju imate je jednostavno isprazniti rječnik pomoću metode **clear()**:

```
sofas_inv.clear()
print(sofas_inv) # you will have an empty dictionary
```

### Kopiranje rječnika

Sjetite se kad smo govorili o kopiranju popisa te smo naglasili da ne možete kopirati jednu stavku u drugu pomoću znaka jednakosti (=) jer će se promjene jednog popisa vršiti na drugom. Slično popisima, rječnici se ne smiju kopirati na taj način. Postoje dva načina da napravite kopiju rječnika.

Prvi je korištenje metode **copy()**:

```
new_inv = sofas_inv.copy()
print(new_inv)
```

Drugi način je korištenje funkcije **dict()**:

```
new_inv = dict(sofas_inv)
```

## 3.4. Naredbe i petlje

U ovom ćemo odjeljku objasniti logiku i upotrebu naredbi i petlji, koji se smatraju bitnim i korisnim znanjem u općim programskim jezicima.

⇒ Naredbe se koriste kroz izraze **if**, **elif**, **else**.

⇒ **For** i **while** se koriste za petlje.

Prije nego što detaljno objasnimo naredbe i petlje, naučit ćemo neke korisne operatore koji se obično koriste u ovim izjavama: booleove izraze (==) i logičke operatore (and, or, not).



### 3.4.1. Booleovski izrazi

Booleov izraz se koristi za određivanje je li izraz ili izjava istinita ili netočna. Postoje dva načina za pisanje booleovskih izraza, prvi koristi `==` operator za usporedbu dviju vrijednosti i vraćanje booleove vrijednosti:

```
print(8 == 8)
```

Ishod:

True

```
print(9 == 8)
```

Ishod:

False

U obje izjave, operator `==` je ocjenjivao jesu li dva cijela broja ista (tj. imaju istu vrijednost). Kao što možete vidjeti, prva izjava je prikazala True, a druga izjava False.

Operator `==` je jedan od mnogih operatora za usporedbu koji se koriste u Pythonu. Ostali operatori usporedbe prikazani su u nastavku:

<code>x &gt; y</code>	x je veći od y
<code>x &lt; y</code>	x je manji od y
<code>x &gt;= y</code>	x je veći ili jednak y
<code>x &lt;= y</code>	x je manji ili jednak y
<code>x != y</code>	x nije jednak y

Vjerojatno ste se već ranije susreli s ovim operatorima, međutim, neki od ovih simbola imaju različite funkcije u Pythonu.

Na primjer, ako želite provjeriti je li neka vrijednost jednaka drugoj, ne biste upotrijebili jedan znak jednakosti (`=`) jer bi to dodijelilo vrijednost varijabli koja se nalazi na lijevoj strani.

Kako biste usporedili vrijednosti u booleovim izrazima, trebate koristiti **dvostruki znak jednakosti (`==`)**.

Također, ako želite provjeriti je li **vrijednost veća ili jednaka (`>=`)** drugoj vrijednosti, prvo morate staviti znak **veće (`>`)** ili **manje (`<`)**, a zatim znak **jednakosti (`=`)**.

Ovo su važne razlike u Pythonu koje biste trebali zapamtiti kako biste izbjegli pogreške ili neželjena iznenađenja.

Pogledajmo neke primjere ovih operatora usporedbe s ključnom riječi **if**:

```
x = 55
y = 500
if y > x:
    print("y is greater than x")
```

U ovom primjeru uspoređujemo varijable *x* i *y* kako bismo provjerili je li *y* veći od *x*. Budući da već znamo da je *y* veći od *x*, odabiremo to ispisati kao ishod.

**\* Uvlačenje koje se pojavljuje u drugom retku (tj. razmak koji se nalazi na početku retka) koristi se da se Pythonu kaže da je ova izjava dio tog određenog skupa kôda. Uvlačenje je važan dio sintakse Pythona kako bi se izbjeglo nastajanje pogrešaka.**

Još jedna ključna riječ koja se koristi u naredbenim izrazima je **elif**, koja se koristi kao alternativa ako prva naredba nije istinita. Pogledajmo primjer operatora usporedbe s ključnom riječi **elif**:

```
x = 55
y = 55
if y < x:
    print("y is greater than x")
elif x == y:
    print("x is equal to y")
```

U ovom primjeru imamo prvu naredbu koja provjerava je li *y* veći od *x*. Budući da nije, program prelazi na drugu naredbu s **elif (kombinacija else i if)** koja provjerava je li *x* jednako *y*, što je točno i ispisuje odgovarajuću izjavu.

Sljedeća ključna riječ koja se koristi u naredbama je **else**, što je posljednja dostupna opcija ako prethodni uvjeti nisu istiniti.

Pogledajmo primjer upotrebe **else** da bismo razumjeli kako funkcionira:

```
x = 500
y = 55
if y > x:
    print("y is greater than x")
elif x == y:
    print("x is equal to y")
else:
    print("x is greater than y")

if x > y: print("x is greater than y")
```

Ovdje možete vidjeti da naredba **if** nije istinita, zatim prelazi na drugu naredbu koja također nije istinita i završava s konačnom dostupnom opcijom u **else**, koja je istinita i ispisuje odgovarajuću izjavu.

Također, ako imate samo kratku naredbu za izvršenje, možete je napisati u samo jednom retku:



```
if x > y: print("x is greater than y")
```

Također možete učiniti istu stvar za kombinaciju **if** i **else** naredbi:

```
print("x is greater than why") if x > y else print("y is greater than x")
```

Druga opcija je uključiti više naredbi **else** u jedan redak:

```
print("X") if x > y else print("=") if x == y else print("Y")
```

### 3.4.2. Logički operatori

Python uključuje 3 logička operatora, kao što smo spomenuli na početku ovog odjeljka, **and**, **or**, i **not**. Ovi operatori imaju analognu upotrebu u prirodnim jezicima koji kombiniraju jedan ili više naredbenih tvrdnji.

Pogledajmo kako se **and** može koristiti s naredbenim tvrdnjama:

```
x = 500
y = 55
z = 800
if x > y and z > x:
    print("Both conditions are true")
```

Pogledajmo primjer operatora **or** koji kombinira naredbene tvrdnje:

```
x = 500
y = 55
z = 800
if x > y and z > x:
    print("One of the two conditions is true")
```

Naš sljedeći primjer fokusiran je na operator **not**:

```
x = 500
y = 55
if not y == x:
    print("x and y are not equal")
```

### 3.4.3. Ugnježdivanje if tvrdnje

Tvrdnje koji sadrže **if** naredbu unutar **drugog if** izraza nazivaju se **ugnježđenim if tvrdnjama**. Pogledajmo sljedeći primjer kako bismo to razumjeli:



```

y = 50
if y > 20:
    print("y is above 20, ")
    if y > 30:
        print("and above 30 ")
    else:
        print("but not above 30")

```

Slično funkcijama, **if** tvrdnje ne bi trebale biti prazne. Međutim, ako iz određenog razloga imate tvrdnju **if** bez ikakvog sadržaja, upotrijebite iskaz **pass** kako biste izbjegli pogreške iz Pythona:

```

x = 500
y = 55
if x > y:
    pass

```

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_ifelse1](https://www.w3schools.com/python/exercise.asp?filename=exercise_ifelse1)

#### 3.4.4. Petlje

Python ima dvije glavne naredbe za stvaranje petlji: **while** i **for**.

Petlja **while** se koristi kada želite nastaviti s izvršavanjem sve dok je naredba istinita. Kao primjer, napravimo **while** petlju koja ispisuje i sve dok je manje od 10:

```

i = 1
while i < 10:
    print(i)
    i += 1

```

U ovom primjeru govorimo Pythonu **sve dok je i manje od 10, nastavi ispisivati i povećavati za 1 svaki put kada se petlja ponavlja**.

**\*Imajte na umu da ako ne povećate i, while petlja će nastaviti raditi bez zaustavljanja i završit ćete u beskonačnoj petlji.**

Također možemo koristiti naredbu **break** ako želimo zaustaviti petlju čak i ako je uvjet istinit:

```

i = 1
while i < 10:
    print(i)
    if i == 5:
        break
    i += 1

```

Još jedna dostupna izjava u petljama je **continue** koja će zaustaviti trenutnu iteraciju i prijeći na sljedeću:

```
i = 1
while i < 10:
    i += 1
    if i == 5:
        continue
    print(i)
```

Naredba **else** koju smo vidjeli u naredbenim tvrdnjama također se može koristiti u petljama:

```
i = 1
while i < 10:
    print(i)
    i += 1
else:
    print("i is no longer less than 10")
```

### Vježba:

[https://www.w3schools.com/python/exercise.asp?filename=exercise\\_while\\_loops1](https://www.w3schools.com/python/exercise.asp?filename=exercise_while_loops1)

Pogledajmo primjer:

Petlje koje koriste ključnu riječ **for** ponavljaju niz stavki koje mogu biti sadržane u popisima, torkama, rječnicima, skupovima ili nizovima.

Pogledajmo primjer ovoga:

```
nouns = ["table", "book", "chair", "house"]
for noun in nouns: # for each noun in the nouns list
    print(noun)
```

U ovoj vrsti petlje ne morate prethodno specificirati varijablu indeksa. Indeksna varijabla je, u ovom slučaju, jednina verzije imenica kao vrijednosti koju treba dohvatiti (tj. svaka stavka na popisu imenica).

Petlja **for** također se može ponavljati kao niz znakova:

```
for x in "book": # for each character in the word book
    print(x)
```

Naredba **break** koju smo vidjeli ranije također se može kombinirati s **for** petljom:

```
nouns = ["table", "book", "chair", "house"]
for noun in nouns: # for each noun in the nouns list
    print(noun)
    if noun == "chair":
        break
```

U ovom primjeru, kada petlja naiđe na imenicu "chair", prestat će ponavljati prije nego što prođe kroz sve stavke na popisu.





Da ste stavili naredbu **break** prije dijela za ispis, što mislite da bi se dogodilo? Isprobajte sami:

```
for noun in nouns: # for each noun in the nouns list
    if noun == "chair":
        break
    print(noun)
```

Još jedna tvrdnja koju smo ranije vidjeli je naredba **continue**, koja prekida trenutnu iteraciju da bi se prešla na sljedeću:

```
for noun in nouns: # for each noun in the nouns list
    if noun == "chair":
        continue
    print(noun)
```

Funkcija raspona je vrlo korisna kada želite odrediti koliko puta želite da se iteracija dogodi. Zadana početna točka funkcije raspona je 0, koja se povećava za 1 pri svakoj iteraciji i završava na određenom broju:

```
for x in range(10):
    print(x)
```

\* Imajte na umu da, budući da raspon počinje od 0, zaustavit će se na broju 9. Ako zapravo želimo da ide do 10, koristili bismo raspon (11).

Imate mogućnost specificiranja početnog i završnog raspona:

```
for x in range(2, 10):
    print(x)
```

Ovdje raspon počinje od 2 i završava na 9, budući da 10 nije uključeno u raspon.

Druga opcija koju imate je da prilagodite za koliko se broj povećava pri svakoj iteraciji:

```
for x in range(2, 40, 2):
    print(x)
```

U ovom primjeru naveli smo raspon koji počinje od 2 do 40 i svaki put se povećava za 2. Stoga će se broj zaustaviti na 38, jer se od 38 nadalje ne može povećati za 2.

U **for** petlji također možete koristiti ključnu riječ **else** kako biste naveli što se događa kada se petlja završi:

```
for x in range(10):
    print(x)
else:
    print("Done!")
```

Pokušajmo upotrijebiti **if** i **else** u **for** petlji:



```
for x in range(10):
    if x == 8: break
    print(x)
else:
    print("Done!")
```

Mislite li da će se naredba **else** izvršiti? Zašto ili zašto ne? Pokušajte saznati!

Sjetite se kada smo govorili o **ugniježdenim if** izjavama, također možete imati ugniježdene petlje:

```
words = ["I", "am", "you", "are", "working"]
nouns = ["book", "house", "person", "love"]
for word in words:           # outer loop
    for noun in nouns:       # inner loop
        print(word, noun)
```

Kao što možete vidjeti, unutarnja petlja će se izvršiti jednom tijekom svake iteracije vanjske petlje.

Slično funkcijama i **if** tvrdnjama, **for** tvrdnje ne bi trebale biti prazne. Međutim, ako iz određenog razloga imate petlju **for** bez ikakvog sadržaja, upotrijebite iskaz **pass** kako biste izbjegli dobivanje pogrešaka iz Pythona:

```
for word in words:
    pass
```

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_for\\_loops1](https://www.w3schools.com/python/exercise.asp?filename=exercise_for_loops1)

### 3.5. Razumijevanje tijeka izvršenja kroz funkcije

Do sada smo vidjeli brojne ugrađene funkcije koje se koriste u Pythonu kao što su `print()`, `type()`, `dict()`, `len()`. Svaka funkcija ima jednu specifičnu svrhu i potrebno je da navedemo koju varijablu želimo da izvrši.

Primjer:

```
x = 5
print(x)
```

U funkciji ispisa moramo u zagradi navesti varijablu `x`, a to je ona koju želimo ispisati. Koja god varijabla ili varijable su sadržane u zagradi nazivaju se **argumenti ili parametri**.

Ova se dva pojma obično koriste naizmjenično, ali **parametri** su varijable navedene unutar zagrada funkcije, npr. `print(x)`, dok se **argumenti** smatraju vrijednostima koje se šalju funkciji kada ju pozovemo.

Iako su ugrađene funkcije Pythona prilično korisne, ne mogu uvijek riješiti sve probleme koje želimo riješiti. Stoga imamo mogućnost kreiranja novih funkcija za rješavanje specifičnih problema, što se smatra jednim od najkorisnijih aspekata jezika opće namjene.

Funkcija sadrži niz tvrdnji koje proizvode željenu operaciju. Sintaksa koja se koristi za funkciju je sljedeća:

```
>> def naziv (argumenti):
    tvrdnje
```

**Def** znači definicija, to je u biti definiranje imena funkcije zajedno s njezinim argumentima kako bi se proizveo željeni rezultat kada se on zatraži. Ugrađene funkcije su već definirane i stoga ne možemo vidjeti tvrdnje koje sadrže, već samo proizvedeni rezultat ili ishod.

Prilikom izrade vlastite funkcije, slobodni ste koristiti bilo koja imena koja želite osim ključnih riječi za Python koje smo spomenuli u prethodnim odjeljcima (odjeljak 3.2.). Argumenti koji se koriste u zagradama funkcije pružaju informacije potrebne za rad funkcije.

Pogledajmo primjer kako bismo razumjeli proces nakon kojeg slijedi funkcija:

```
def my_function():
    print("Hello, World!")
```

Ovdje smo stvorili funkciju bez ikakvih potrebnih argumenata koja će ispisati tekst naveden u drugom retku kôda.

Sjećate li se kada smo u prethodnom odjeljku govorili o uvlačenju? Ovdje se primjenjuje ista logika gdje uvlačenje označava da je kôd koji treba izvršiti **lokalan** za funkciju.

Stoga bi se pisanje tvrdnje `ispisa("Hello, World!")` izvan funkcije kao što smo to učinili u prethodnim primjerima, smatralo **globalnom** naredbom. Međutim, sada kada je napisano unutar funkcije, smatra se lokalnom.

Za pozivanje funkcije jednostavno koristimo njezino ime praćeno zagradama:

```
my_function()
```

Ishod ove funkcije bit će:

```
Hello, World!
```

Pogledajmo još jedan primjer s navedenim argumentom unutar zagrada funkcije:



```
def my_function(country):
    print("I am from " + country)
```

Kada se funkcija pozove, naziv zemlje se prosljeđuje unutar funkcije koja će biti ispisana prema uputama. Zemlja argumenta može se navesti kada pozovemo funkciju:

```
my_function("France")
my_function("Greece")
my_function("Germany")
```

Ishod:

```
I am from France
I am from Greece
I am from Germany
```

Također možete odrediti tip podataka argumenta koristeći sljedeću sintaksu:  
argument:datatype

Primjer:

```
def my_function(country:str):
    print("I am from " + country)
```

Prilikom pozivanja funkcije, Python će očekivati da zemlja bude u obliku niza. Ako nije, pojavit će se pogreška.

```
my_function("France")
my_function(France) #will raise error
```

Osim toga, možete imati više od 1 argumenta. Recimo da smo htjeli navesti grad i državu:

```
def my_function(city, country):
    print("I am from " + city + ", "+ country)
```

Imajte na umu da smo uključili navodnike sa zarezom kako bismo odvojili grad i državu jedan od drugog kako se dvije riječi ne bi ispisivale bez razmaka. Budući da smo naveli 2 argumenta, moramo pozvati točno broj argumenata naveden u zagradi, ne više ili manje, da bi funkcija radila. U suprotnom će se pojaviti pogreška.

```
my_function("Paris", "France")
```

Ishod:

```
I am from Paris, France
```

```
my_function("Paris") # this will raise an error in Python
```

Druga opcija koja postoji, ako ne znate broj argumenata koji će biti uključeni u funkciju, jednostavno dodajte zvjezdicu (\*) ispred naziva argumenta u definiciji funkcije.



```
def my_function(*countries):
    for country in countries:
        print("I have visited ", country)
```

Na taj način funkcija će primiti tuple argumenata i stavkama se može pristupiti u skladu s tim.

**\* Ove vrste nepoznatog broja argumenata nazivaju se proizvoljnim argumentima i često se nazivaju \*args u dokumentaciji Pythona.**

Ovdje smo naveli 3 zemlje koje treba ispisati u zasebnim rečenicama i u jednoj rečenici:

```
my_function("Ireland", "France", "Belgium")
my_function("Ireland, France, Belgium")
```

Ishod:

I have visited Ireland

I have visited France

I have visited Belgium

I have visited Ireland, France, Belgium

Druga opcija koja vam je dostupna je slanje argumenata kao parova key = value (tipka=vrijednost). Na taj način redoslijed nije bitan.

Primjer:

```
def my_function(first, last):
    print("My name is " + first + last )
my_function(first = "Jane ", last = "Kent")
```

Još jedna dostupna opcija je korištenje **dvije zvjezdice (\*\*)** ispred naziva argumenta u definiciji funkcije ako ne znate koliko argumenata ključne riječi treba proslijediti. Na taj će način funkcija primiti argumente u obliku rječnika i pristupiti im u skladu s tim:

```
def total_words(**words):
    print(words, type(words))
total_words(paper = 6, I = 10, nice = 9)
```

Kao što vidite, možete dodati nove argumente kada pozovete funkciju. Imajte na umu da se ove vrste argumenata nazivaju **argumentima proizvoljnih ključnih riječi** i često se nazivaju **\*\*kwargs** u dokumentaciji za Python.



Nadalje, još jedna opcija koju imate je postavljanje zadane vrijednosti parametra. To znači da ako funkciju pozovemo bez argumenta, ona će koristiti zadanu vrijednost koju smo postavili:

```
def my_function(country= "United Kingdom"):
    print("I am from" + country)

my_function("France")
my_function("Malta")
my_function()
```

Argument može dopustiti bilo koju vrstu podataka kao argument (npr. niz, popis, rječnik, itd.). Funkcije također mogu vratiti vrijednosti korištenjem naredbe **return**:

```
def calc(x):
    return 3 * x

print(calc(5))
print(calc(8))
print(calc(9))
```

Budući da nismo specificirali funkciju ispisa u našoj kreiranoj funkciji, moramo je koristiti kada pozivamo funkciju kako bismo dobili rezultat.

Općenito, funkcije ne bi trebale biti prazne. Međutim, ako iz određenog razloga imate definiciju funkcije bez ikakvog sadržaja, koristite naredbu `pass` kako biste izbjegli dobivanje pogrešaka iz Pythona:

```
def my_function():
    pass
```

**Vježba:** [https://www.w3schools.com/python/exercise.asp?filename=exercise\\_functions1](https://www.w3schools.com/python/exercise.asp?filename=exercise_functions1)

### 3.6. Shvaćanje sastavljanja dijelova – Kako napraviti program

Do sada smo naučili mnogo različitih tvrdnji, ključnih riječi, metoda i funkcija koje se koriste u Pythonu. Sada se možda pitate, kako zapravo možemo napraviti jednostavan program? Proći ćemo kroz proces izrade malog programa korak po korak, tako da možete razumjeti kako koristiti ono što smo naučili.

Želimo izraditi program koji:

1. čita tekstualne dokumente,
2. stvara rječnik koji sadrži sve riječi iz tekstualnog dokumenta i njihove učestalosti,
3. dopušta korisnicima pisanje riječi i prikaz njihovih učestalosti,
4. osigurava informativnu poruku ukoliko riječ ne postoji.



Ovo bi se u početku moglo činiti previše, ali ne brinite! Proći ćemo kroz to korak po korak.

Ovdje ćemo naučiti kako kreirati rekurzivne funkcije, što znači da će se funkcije međusobno pozivati kako bi izvršile naš program.

Prvo, trebamo pozvati modul, koji se zove **string**. Ugrađeni string modul sadrži nekoliko funkcija koje vam omogućuju manipuliranje nizovima u Pythonu, koje ćemo koristiti za uklanjanje svih skupova interpunkcije u kasnijoj upotrebi.

U Pythonu su dostupni mnogi moduli za razne svrhe koje možete pozvati jednostavnim korištenjem ključne riječi import:

```
import string
```

Prva funkcija će pročitati tekstualnu datoteku i stvoriti rječnik:

```
def process_text(filename):
    dictionary = dict()
    fin = open(filename, 'r')
    for line in fin:
        process_line(line, dictionary)
    return dictionary
```

Uvijek pokušavamo dati ime funkciji koje možemo razumjeti, a argument naziva datoteke bit će naveden kada ga pozovemo. Sljedeći redak stvara prazan rječnik. **Lokalna varijabla fin** poziva funkciju **open** da otvori datoteku i pročita je. Petlja **for** u sljedećem retku poziva sljedeću funkciju koja će obraditi svaki redak, kreirati naš rječnik i vratiti ga.

Možda zvuči pomalo tehnički, ali prilično je jednostavno. U biti smo stvorili funkciju koja će stvoriti prazan rječnik, pročitati naziv datoteke i obraditi svaki redak datoteke kako bi se stvorio rječnik.

Sada, zašto želimo obraditi svaki redak?

Zato što će Python raditi razliku između velikih i malih riječi, kao i interpunkcijskih točaka pored riječi, čak i ako je riječ ista kao što su "Love" i "love" ili "love" i "love".

Budući da želimo prebrojati učestalost svake riječi u datoteci, moramo se pobrinuti da su sve riječi mala slova i da su sve interpunkcijske točke (tj. zarezi, točke, uskličnici itd.) uklonjene kako bi program ispravno brojao.

Napišimo funkciju koja obrađuje svaki redak naše datoteke:



```
def process_line(line,dictionary):
    line = line.replace('-', '')

    for word in line.split():
        word = word.strip(string.punctuation + string.whitespace)
        word = word.lower()
        dictionary[word] = dictionary.get(word,0) +1
```

Opet, ovdje koristimo **for** petlju za ponavljanje kroz svaki redak teksta.

⇒ Najprije, crtice zamjenjujemo razmakom jer se ne mogu ukloniti pomoću funkcije **string.punctuation**.

⇒ Nakon što zamijenimo crtice, počinjemo s dijeljenjem retka u zasebne riječi i uzimamo svaku riječ kako bismo uklonili interpunkcijske znakove, razmake, a zatim je napisali malim slovima.

⇒ Nakon cijele obrade riječ se dodaje u rječnik i **ako riječ postoji dodajete 1, a ako ne, dodajete 0**.

Do sada smo kreirali dvije funkcije koje čitaju, uređuju tekst i stvaraju rječnik učestalosti riječi. U ovom trenutku dodjeljujete rječniku varijabli funkciju koja obrađuje tekst, gdje specificirate naziv tekstualne datoteke (tj. 'the\_veldt.txt'). Sve funkcije su povezane na temelju varijabilnog rječnika koji će sadržavati sve riječi koje se nalaze u zadanom tekstu s njihovim učestalostima.

```
dictionary = process_text('the_veldt.txt')
```

Sada smo gotovi s 2 od 4 akcije koje želimo da program obavi, preostale su nam još samo 2.

Sada želimo da korisnici mogu napisati riječ i ako riječ postoji u tekstu da prikaže njezinu učestalost, u protivnom želimo obavijestiti korisnika da riječ ne postoji.

Napišimo funkciju koja će usporediti unos našeg korisnika s rječnikom koji smo kreirali:

```
def findwords(dictionary):
    for key,value in dictionary.items():
        if key == find_word:
            return(value)
    return("This word was not found in the text, please look for another word ")
```

Ova funkcija se koristi za pristup ključu i vrijednosti svake stavke rječnika kroz **for** petlju. Ako je ključ (riječ) isti kao i riječ koju je napisao korisnik, onda vraća vrijednost (tj. učestalost dane riječi). U suprotnom, vraća izjavu da riječ nije pronađena i poziva korisnika da potraži drugu.



```
while True:
    find_word = input("Please enter word to find its frequency, or type 'q' to quit: ").lower()
    if find_word != 'q':
        print(findwords(dictionary))
        continue
    else:
        break
```

Sada za zadnji dio kôda, želimo da se program ponavlja sve dok korisnik ne odredi prestanak. Za ovaj dio koristit ćemo **while** petlju koja nije u funkciji.

Jednostavno koristimo **while True** petlju kako bi program I dalje radio. Budite oprezni kada koristite **while True** petlju jer možete završiti u beskonačnoj petlji. **While True** označava sve dok je to istina, te nastavlja s izvršavanjem programa.

Dodjeljujemo varijablu za korisnički unos kako bismo je mogli usporediti s tipkama rječnika (riječima) te sve što korisnik unese bit će pretvoreno u mala slova kada ga program pročita. Razlog tome je taj što korisnik može napisati riječ svim velikim slovima i program neće dati rezultat, jer Python ne prepoznaje da je riječ o istoj riječi, kako smo već pojasnili da je Python osjetljiv na velika i mala slova.

Sljedeći red kaže da ako unos nije jednak 'q', onda ispišite rezultate funkcije koja dohvaća učestalost i prijedite na sljedeću iteraciju. Inače, ako je unos korisnika **q**, on zaustavlja program (tj. prekida petlju).

Ovisno o tome tko piše kôd, ovaj mali program mogao je biti napisan na različite načine. Pokušali smo uključiti što više stvari koje smo ovdje naučili, kao i uvesti rekurzivne funkcije i kako razbiti vaš program na male i upravljive dijelove kôda.

**\* Imajte na umu da kada pišete kôd, trebate pokušati pokrenuti svaki mali dio kôda kako biste uhvatili potencijalne pogreške prije nego što nastavite na sljedeći dio.**

Nakon što dovršite svoj program, možda ćete se morati vratiti i urediti kako biste uklonili neke eksperimentalne/početne dijelove kôda ili konsolidirali više iskaza kako biste program učinili kompaktnijim i lakšim za čitanje.

### 3.7. Kako prilagoditi program svojim potrebama

Vidjeli smo kako možemo napraviti mali program u Pythonu, međutim, ponekad možete naići na programe koji su već napisani i možda ćete morati napraviti neke prilagodbe kôda

kako biste poboljšali ili promijenili krajnji rezultat.

Ovisno o tome tko je napisao kôd i koliko je dugačak, ovo može biti težak i zahtjevan postupak.

Možete slijediti istu logiku kao kada gradite program, gdje uzimate male dijelove kôda da shvatite što svaka funkcija radi, ako nije jasno.



Također, važno je pisati komentare kada radite s kôdom jer komentari mogu biti od velike pomoći drugim ljudima koji čitaju vaš kôd, ali i vama.

Dakle, što učiniti kada nema komentara na program?

Prije svega, pokrenite program da vidite kakav je njegov ishod i primijetite knjižnice ili module koji su uvezeni. Zatim biste trebali potražiti početnu točku kôd i pokušati razumjeti tijek izvršenja. Još jedan koristan savjet bio bi pogledati varijable koje program sadrži i njihovu upotrebu u cijelom programu. Nakon što ste dobro razumjeli kako program radi, možete početi uređivati.

**\* Budite oprezni kada pokušavate urediti tuđu funkciju jer biste je mogli pokvariti.**

Dobra praksa pri prilagođavanju programa nije korištenje određenog broja varijabli, već korištenje **\*args** i **\*\*kwargs** kako bi vaš kôd bio fleksibilniji i prilagodljiviji kada bi ga drugi korisnik trebao prilagoditi. Ranije smo u modulu vidjeli korištenje ove dvije varijable.

Podsjetimo se, **\*args** vam omogućuje prosljeđivanje različitog broja pozicijskih argumenata, a **\*\*kwargs** vam omogućuje prosljeđivanje različitog broja ključnih riječi. Važnost oba je upotreba zvjezdice(a) (\* ili \*\*), dok njihova imena mogu biti koja god želite.

Za još više savjeta i primjera pogledajte sljedeće web stranice:

- freeCodeCamp - <https://www.freecodecamp.org/news/args-and-kwargs-in-python/>
- DZone - <https://dzone.com/articles/adding-functionality-to-legacy-code>
- Codecademy - <https://www.codecademy.com/resources/blog/how-to-work-with-code-written-by-someone-else/>

### 3.8. Pogreške sintakse, vremena izvođenja i semantičke pogreške – rukovanje pogreškama u Pythonu

Postoje tri vrlo važne različite vrste pogrešaka koje se mogu pojaviti prilikom pisanja programa i korisno je znati njihove razlike i kako ih rano uočiti:

1. **Sintaktičke pogreške** nastaju kada Python prevodi izvorni kôd u binarni oblik i ukazuje da je neki dio sintakse pogrešan, npr. uvlačenje, nedostajanje dvotočke na kraju izraza def ili nedostajanje zagrada. Slično prirodnim jezicima, kada koristimo nispravnu sintaksu u Pythonu, dobit ćemo pogrešku koja kaže: nevažeća sintaksa.

Sintaktičke greške lakše je uočiti, te evo nekoliko stvari na koje treba obratiti pažnju:

- Korištenje ključne riječi Python za naziv varijable
- Nedostaje dvotočka (:) iza **for**, **while**, **if** i **def** naredbi
- Uvlačenje unutar petlji i naredbi
- Podudarni navodnici za nizove, tj. dvostruki ili pojedinačni



- Provjerite jeste li propustili nijedan navodnik kada pišete nizove
- Korištenje jednog znaka jednakosti umjesto dva u naredbenoj tvrdnji
- Nezatvorena zagrada u bilo kojem retku vašeg kôda, tj. ), ], }

Ako ne možete pronaći sintaktičku pogrešku, stvorite novu datoteku i dodajte svoj kôd redak po redak od početka.

2. **Pogreške tijekom izvođenja** nastaju kada se primjenjuje sljedeće: 1) program ne radi ništa, 2) program ulazi u beskonačnu petlju ili rekurziju, 3) dobijete pogrešku iznimke ili 4) ako ste dodali previše naredbi za ispis.

Pogledajmo neke načine na koje možete prevladati ili uočiti korijen za sve te slučajeve:

- Kada vaš program ne radi ništa, provjerite jeste li ga pozvali da se izvrši u interaktivnoj konzoli
- Ako uđete u beskonačnu petlju, zaustavite program i dodajte izjave za ispis tamo gdje mislite da bi mogao biti problem i pokušajte koristiti ljestve (#) ispred dijelova kôda da vidite što se događa
- Pogreške iznimke spadaju u 5 glavnih kategorija:
  - NameError** gdje pokušavate koristiti varijablu koja ne postoji, tj. lokalne varijable;
  - TypeError** se može pojaviti iz više razloga zbog netočne upotrebe vrijednosti, nepodudarnosti između stavki u format niza i konvertiranih stavki ili pogrešnog broja argumenata;
  - KeyError** se može dogoditi kada pokušate pristupiti stavci u rječniku pomoću ključa koji ne postoji;
  - AttributeError** se može dogoditi kada pokušavate pristupiti atributu koji ne postoji;
  - IndexError** gdje ne postoji usklađenost između indeksnog broja popisa, niza ili torke i njegove duljine.

3. **Semantičke pogreške** su obično pogreške koje je najteže odgonetnuti budući da je program pokrenut, ali ne daje željeni ishod. Možda ste mislili da redoslijed kojim stavljate svoje tvrdnje ima smisla, međutim, Python bi se mogao ponašati na neočekivane načine.

Kada naiđete na semantičke pogreške, korisno je učiniti sljedeće:

- Rastavite kôd na manje dijelove kako biste shvatili kako se program ponaša u svakom koraku



- Preispitajte svoje misli ili bilješke i logiku iza svakog retka kôda
- Koristite izjave za ispis da vidite što program radi
- Ako koristite duge tvrdnje, koristite privremene varijable da provjerite vrste varijabli
- Dodijelite varijablu svom izrazu prije povratne tvrdnje
- Ako ne možete otkriti pogrešku, napravite kratku stanku ili zatražite pomoć.

### 3.9. Vježba

Jedan od najvažnijih aspekata kodiranja je praksa. Što više vježbate, to ćete postati bolji. Nadamo se da ćete do kraja ovog modula dobro razumjeti logiku pisanja kôda, različite vrste podataka i njihovu upotrebu, kao i kako izgraditi male programe pomoću funkcija.

Potičemo vas da vježbate što je više moguće kako biste pomoću kodiranja kreirali vlastite individualne projekte i poboljšali izgled i uspjeh u karijeri.

#### Spremni za početak vježbanja? Sretno kodiranje!

Ovdje možete pronaći popis različitih web-mjesta koje možete koristiti za daljnje razvijanje vještina kodiranja:

- ⇒ **W3Schools** - [https://www.w3schools.com/python/python\\_exercises.asp](https://www.w3schools.com/python/python_exercises.asp)
- ⇒ **GeeksforGeeks** - <https://www.geeksforgeeks.org/python-exercises-practice-questions-and-solutions/>
- ⇒ **PYnative** - <https://pynative.com/python-exercises-with-solutions/>

#### Reference:

Downey, A. Elkner, J. & Meyers, C. (2008). How to Think Like a Computer Scientist: *Learning with Python*. Green Tea Press: Wellesley, Massachusetts.

GeeksforGeeks (2021). *Top 10 Python IDE and Code Editors in 2020*. <https://www.geeksforgeeks.org/top-10-python-ide-and-code-editors-in-2020/>

Karpinski, Z., Biagi, F., & Di Pietro, G. (2021). Computational Thinking, Socioeconomic Gaps, and Policy Implications. IEA Compass: Briefs in Education. 12. *International Association for the Evaluation of Educational Achievement*.

O' Dea, B. (2021). *Python named most in-demand coding language for 2022*. <https://www.siliconrepublic.com/careers/python-most-in-demand-coding-language-2022>

Programiz. *How to Get Started with Python?* <https://www.programiz.com/python-programming/first-program>

Real Python. *Python args and kwargs: Demystified*. <https://realpython.com/python-kwargs-and-args/>

W3Schools. *Python Tutorial*. <https://www.w3schools.com/python/>



## DIO B: MODUL CodER Mikrokontrolera (10 sati)

### Opis:

Ovaj dio pruža uvod u mikrokontrolere, njihovu upotrebu i primjenu na temelju primjera iz stvarnog života. Prvo potpoglavlje objašnjava komponente mikrokontrolera i njegove različite vrste kako bi se učenici upoznali s ovim konceptom. Zatim se prelazi na korištenje Arduino softvera i na način na kako se povezati s Arduino mikrokontrolerima u pristupu korak po korak.

### Vrijeme potrebno za rad:

10 sati

### Ishodi učenja:

Do kraja ovog tečaja polaznici će moći:

- ⇒ Prepoznati što je mikrokontroler i biti u stanju identificirati različite vrste mikrokontrolera
- ⇒ Znati razliku između analognog i digitalnog ulaza/izlaza (I/O)
- ⇒ Koristite osnovnu sintaksu Arduino IDE
- ⇒ Izvršiti različite primjere Arduino IDE i mikrokontrolera

### Potreban materijal I sredstva:

- ⇒ Računalo ili prijenosno računalo
- ⇒ Pristup internetu Internet Access
- ⇒ Arduino Uno
- ⇒ Arduino IDE

### Praktičnosti:

Ovaj dio modula predviđen je za 10 sati učenja. Svaki dio modula ima određeno vrijeme, ali učenik ili edukator/trener može slobodno odlučiti koliko će vremena potrošiti na svaku podtemu ovisno o predznanju i angažmanu u sličnim temama. Sadržaj se temelji na progresivnom razvoju i služi za pružanje osnovnih znanja i vještina početnicima.

### Potpoglavlja:

1. Uvod u mikrokontrolere



2. Osnove programiranja s Arduinoom
3. Primjena Arduina

## 1. Uvod u mikrokontrolere

- ⇒ **Broj sudionika:** 1-10 po voditelju
- ⇒ **Trajanje:** 1.5 sati
- ⇒ **Metode podučavanje:** prezentacija, vođene instrukcije, iskustveno učenje
- ⇒ **Potrebni materijali:** prezentacija, Arduino ploče i stabilna internet veza

### 1.1. Što je mikrokontroler

Prije nego što objasnimo što je mikrokontroler, razmotrimo sljedeća pitanja:

- Jeste li ikada bili znatiželjni o tome kako rade gadgeti? Koja je logika iza njih?
- Jeste li ikada željeli znati kako funkcioniraju sustavi koji kontroliraju dizala ili elektroničke igračke?
- Ili čak stvoriti vlastitog robota ili elektroničke signale za model željeznice?
- Jeste li se ikada zapitali kako se podaci o vremenu prikupljaju i analiziraju?

Jeste li znatiželjni? Pa, krenimo onda!

Mikrokontroleri mogu olakšati bolje razumijevanje ovih elektroničkih procesa kroz praktične aktivnosti.

Mikrokontroler je kompaktni integrirani krug odgovoran za izvršavanje određene funkcije u uređaju. On interpretira podatke koje prima od svojih ulaznih i izlaznih (I/O) perifernih uređaja preko svog središnjeg procesora.

Mikrokontroleri se mogu naći u raznim sustavima i uređajima. Neke primjene mikrokontrolera mogu se naći u kamerama, kontrolama motora, bravama vrata, požarnim ili dimnim alarmima ili sensorima temperature, svjetla i boje.

Razmotrimo primjer automobila s mnogo mikrokontrolera za upravljanje pojedinačnim sustavima kao što su svjetlosni senzori, protublokirajući kočioni sustavi, električni prozori ili kontrole kočnica. Vozilo može imati čak 50 mikrokontrolera odgovornih za određene operacije koji komuniciraju međusobno ili s drugim složenijim sustavima za obavljanje odgovarajućih radnji.

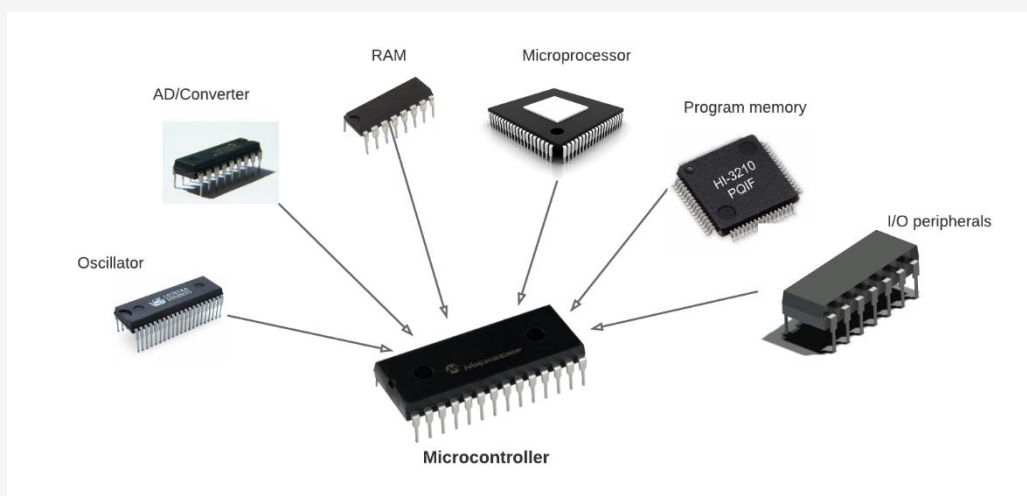
Mikrokontroler je u suštini mali čip koji se sastoji od sljedećih elemenata:

1. **Procesor (CPU):** Procesor ili središnja procesorska jedinica (CPU) koristi se za obradu i izvršavanje različitih instrukcija koje usmjeravaju funkciju mikrokontrolera. On čita i izvodi osnovne aritmetičke, logičke i I/O operacije. Također je odgovoran za prijenos



podataka za komunikaciju naredbi drugim komponentama unutar većeg ugrađenog sustava.

2. Memorija: Glavna funkcija memorije mikrokontrolera je pohranjivanje podataka kako bi ih poslali procesoru i odgovorili na njegovu unaprijed određenu funkciju programiranja. Mikrokontroler ima dvije glavne vrste memorije:
  - a. Programska memorija je ona koja pohranjuje dugoročne informacije o operacijama za čije je izvršenje mikrokontroler programiran. Ova vrsta memorije ne treba izvor napajanja za rad i pohranjuje informacije na duži vremenski period.
  - b. Memorija podataka ili Random Access Memory (RAM) koristi se za pohranjivanje privremenih podataka tijekom izvođenja programa. Ova vrsta memorije čuva podatke sve dok je uređaj spojen na izvor napajanja.
  
3. Ulazno/izlazni (I/O) periferni uređaji: U/I periferije su odgovorne za komunikaciju mikrokontrolera s drugim komponentama. Ulazni portovi primaju informacije i šalju ih na daljnju obradu procesoru kao binarne podatke. Na temelju naredbi procesora, izlazni portovi izvršavaju potrebne zadatke.



**Slika 17** – Elementi mikrokontrolera

Izvor: <https://www.circuitbasics.com/introduction-to-microcontrolleres/>

Na tržištu postoji mnogo primjera mikrokontrolera. Arduino, Scratch, Microbit i Raspberry PI su među najpopularnijim.

Evo njihovih službenih web-mjesta da ih provjerite u svoje vrijeme:

- Arduino: <https://www.arduino.cc/en/software>
- Scratch: <https://scratch.mit.edu/>
- Microbit: <https://microbit.org/>

- Raspberry PI: <https://www.raspberrypi.org/>

U ovom modulu ćemo se fokusirati na Arduino mikrokontrolere.

## 1.2. Što je Arduino i njegove različite vrste

### Što je Arduino

Arduino je platforma otvorenog izvora koja se koristi za izgradnju elektroničkih projekata. Arduino se sastoji od fizičke programibilne pločice (često se naziva mikrokontroler) i dijela softvera ili IDE-a (Integrirano razvojno okruženje) koji radi na računalu, a koristi se za pisanje i prijenos računalnog kôda na fizičku ploču, koja je odlična platforma za izradu projekata prototipova i kreacija (Green Steam Incubator, 2019.).

Arduino ploče imaju način za razumijevanje elektroničkih procesa kroz praktične aktivnosti. Ovaj sustav su stvorili Massimo Banzi i David Cuartielles 2005. godine. On nudi alternativu inače skupim mikrokontrolerima i omogućuje izgradnju interaktivnih projekata kao što su GPS sustavi za praćenje, svjetlosni senzori, roboti na daljinsko upravljanje i elektroničke igre.

Glavne komponente Arduina su:

1. Softver: Arduino IDE je odgovoran za pisanje programa koji se koriste za komunikaciju s vašim hardverom. Funkcije ploče se kontroliraju na temelju uputa koje se šalju mikrokontroleru putem Arduino IDE.
2. Hardver: Arduino ploča ima različitih vrsta i mogu čitati analogne ili digitalne ulazne signale s različitih senzora kako bi proizvele ishod. Neki primjeri izlaza uključuju aktiviranje motora, uključivanje/isključivanje LED-a ili zaključavanje/otključavanje vrata.
3. Programski jezik: Programski jezik koji se koristi u Arduinu je pojednostavljena verzija C++.

### Vrste Arduino ploča

Dostupno je nekoliko vrsta Arduino ploča ovisno o mikrokontroleru koji se koristi. Razlike su uglavnom izražene u sljedećim značajkama:

- broj ulaza i izlaza (npr. senzori, LED diode, tipke na ploči)
- brzina
- radni napon
- faktor oblika itd.

Neke su ploče dizajnirane isključivo za ugradnju i ne nude sučelje za programiranje. Drugi se mogu izravno napajati iz baterije od 3,7 V, dok drugima treba najmanje 5 V za rad. Bez obzira na njihove razlike u ovim značajkama, još uvijek se mogu programirati kroz Arduino IDE.

Nekoliko primjera različitih tipova Arduino ploča može se vidjeti na donjoj slici.



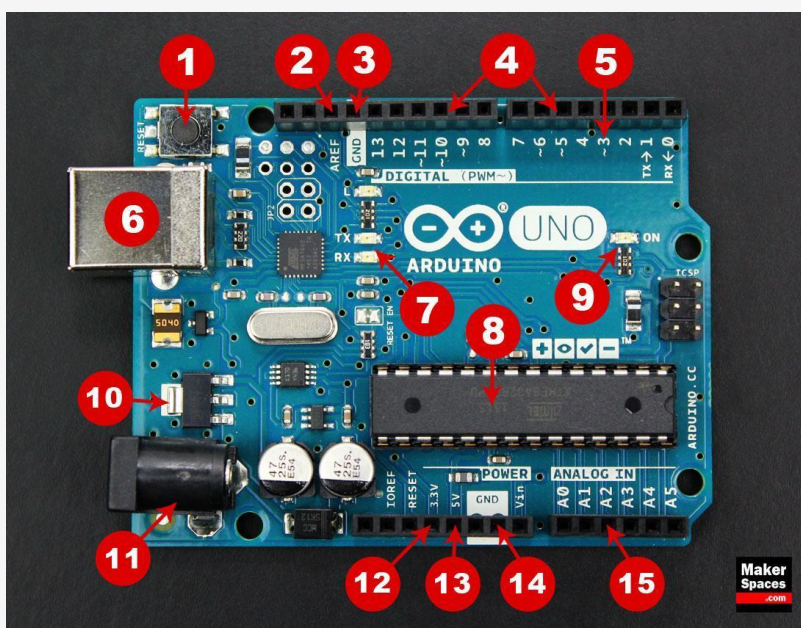




Slika 18 – Različite vrste Arduino ploča

Izvor: <https://techatronic.com/types-of-arduino-boards-arduino-uno-mega-mini-specification/>

Među najpopularnijim Arduino pločama je Arduino Uno. Iako nije bila prva ploča koja je puštena na tržište, i dalje ostaje jedna od najaktivnijih i najšire dokumentiranih ploča.



Slika 19 – Arduino UNO.

Izvor: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

1. Gumb za resetiranje – Ponovno pokreće bilo koji kôd koji je učitao na Arduino ploču
2. AREF – Označava "Analognu referencu" i postavlja vanjski referentni napon

3. Pin za uzemljenje (GND) – Zatvara električni krug i pruža potpunu zajedničku referencu
4. Digitalni ulaz/izlaz – Pinovi 0-13 mogu se koristiti za digitalni ulaz ili izlaz
5. PWM – igle označene simbolom (~) mogu simulirati analogni izlaz
6. USB veza – Napaja vaš Arduino i učitava skice
7. TX/RX – Odašilje i prima naznaku podataka od LED dioda
8. ATmega mikrokontroler – pohranjuje programe (mozak Arduino)
9. LED indikator napajanja – svijetli kada je ploča priključena na izvor napajanja
10. Regulator napona – Kontrolira količinu napona koji ide u Arduino ploču
11. DC Power Barrel Jack – Napaja vaš Arduino pomoću napajanja
12. Pin od 3,3 V – napaja vaše projekte od 3,3 volta
13. Pin od 5V – napaja 5 volti za vaše projekte
14. Igle za uzemljenje – Zatvara električni krug i osigurava kompletnu zajedničku referencu
15. Analogni pinovi – očitava signal s analognog senzora i pretvara ga u digitalni

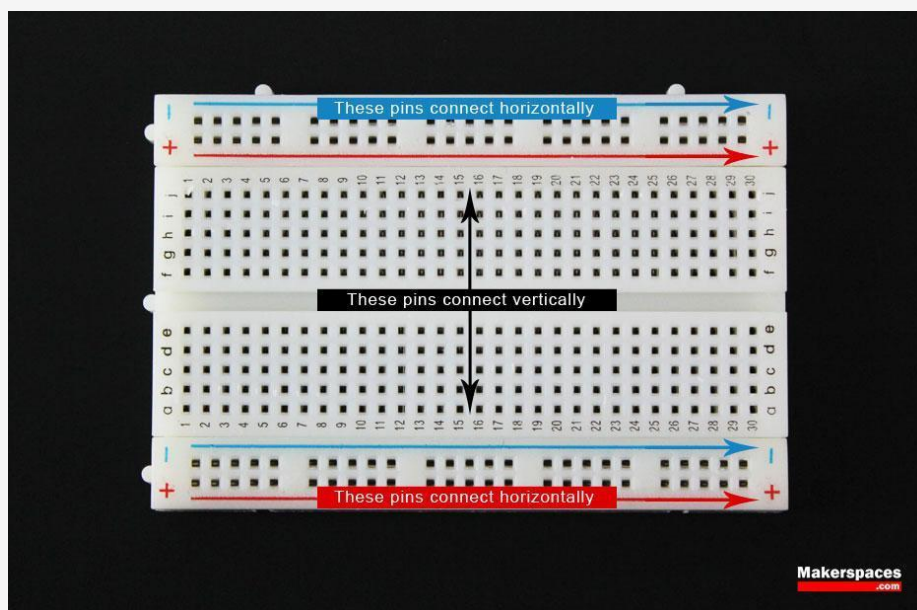
Arduino Uno ploče moraju biti spojene na izvor napajanja da bi radile. Postoje različiti načini za povezivanje ploče s izvorom napajanja, kao što je izravno preko računala putem USB-a ili u slučaju mobilnih projekata preko 9V baterije. Posljednja metoda zahtijeva korištenje 9V AC napajanja za rad.



**Slika 20** – Izvori napajanja za Arduine

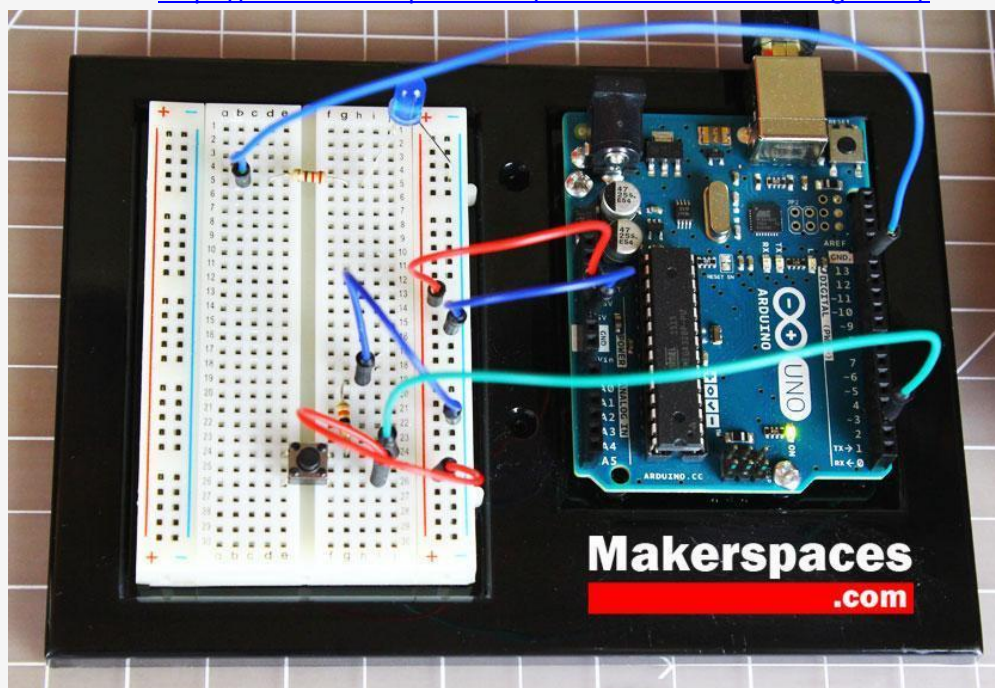
Izvor: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

Još jedna važna komponenta Arduino ploče je matična ploča, također nazvana *Solderless Breadboard*. Koristi se u fazi izrade prototipa za procjenu funkcionalnosti sklopa i omogućuje privremeno stvaranje i eksperimentiranje različitih dizajna sklopova. Spojne točke (rupe) plastičnog kućišta sadrže metalne kopče povezane jedna s drugom trakama od provodljivog materijala. Međutim, matična ploča se ne napaja sama i treba je spojiti na Arduino ploču preko kratkospojnih žica. Također, ove žice se koriste za formiranje kruga spajanjem otpornika, prekidača i drugih komponenti zajedno.



Slika 21 – Breadboard

Izvor: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>



Slika 22 – Dovršeni Arduino krug

Izvor: <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>

### 1.3. Koncepti: ulaz, izlaz, analogni, digitalni

#### Ulazi i izlazi (I/O) unutar konteksta programiranja

Ulazi i izlazi predstavljaju interakcije između robota/stroja i stvarnog svijeta. Jednostavnijim riječima, ulazi su podaci koje robot/stroj prima, a izlazi su rezultati koje možemo vidjeti.

Ulazi se obično prenose sensorima kao što su prekidači, potenciometri ili kamere, dok se izlazi odnose na trenutne radnje koje iniciraju motori, kao što je paljenje LED-a ili paljenje alarma.

Razmotrite sljedeći primjer: Tipke na tipkovnici predstavljaju ulaze. Tijekom tipkanja, računalo prima poredane informacije. Međutim, taj proces nama nije vidljiv. Računalo ili stroj preuzima ulaz (tj. sve što je upisano) i ispisuje ga na ekran.

U kontekstu Arduina: Program može uzeti gumb kao ulaz, koji će upaliti LED svjetlo ako je aktiviran. Informacije koje obrađuje tipka nisu vam vidljive, ali LED svjetlo koje se pali ili gasi dokaz je izlaza.

Postoje dvije vrste ulaza/izlaza: analogni i digitalni I/O.

#### Koja je razlika između analognog i digitalnog ulaza/izlaza

Postoje dva načina za razlikovanje analogni signal od digitalnog signala:

1. Po tipu senzora
2. Po načinu obrade (tj. vremenu i rezoluciji)

1. Analogni naspram digitalnih senzora: LED može biti ON/OFF ili imati promjenjiv intenzitet, kao što je ON s niskim intenzitetom ili ON s visokim intenzitetom.

⇒ Ako je senzor prekidač postavljen na LED, on će kontrolirati hoće li LED biti uključen ili isključen i neće detektirati ništa drugo. Ovo je **digitalni ulaz**.

⇒ Ako je senzor LDR (otpornik ovisan o svjetlu), otkrit će svjetlo i pretvorit će ga u analognu vrijednost koja se nalazi između 0-255. Možemo otkriti je li LED na 0% (isključeno) ili na 100% (puna svjetlina) i također očitati mnoge druge vrijednosti između (npr. 20, 23%, 86%). Ovo je **analogni ulaz**.

2. Analogna naspram digitalna obrada: Da biste utvrdili je li analogna ili digitalna, jednostavno provjerite njihovu obradu na temelju:

⇒ Vremena: Analogni kontinuirano obrađuje informacije i kad god se unos promijeni, izlaz se trenutno mijenja prema ulazu. Ažuriranje je trenutno. Za digitalni, obrada će imati kratku odgodu dok sustav ne registrira promjenu. Ovo kašnjenje se utvrđuje "frekvencijom uzorkovanja" i kontrolira satom.

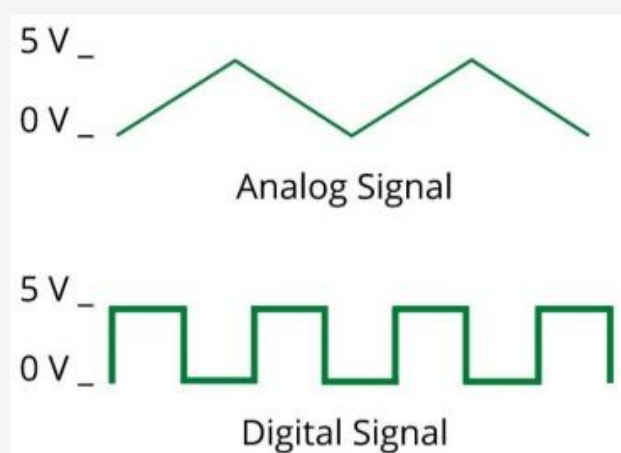
⇒ Rezolucije: Analogna obrada ima beskonačne rezolucije jer nikada ne prestaje s obradom i ima mnogo različitih vrijednosti. Nasuprot tome, digitalna



razlučivost radi s binarnim brojevima, što znači da obrađuje samo dvije vrijednosti i signale.

Primjeri s LED svjetlom:

- Uključite ga/isključite s analognom obradom: kada je dimer aktiviran, krug će u skladu s tim odmah promijeniti intenzitet LED-a.
- Uključite/isključite ga digitalnom obradom: sa zadanom učestalosti uzorkovanja, svakih 5 sekundi, sustav će očitati prekidač i pokazati je li svjetlo uključeno ili isključeno, bez obzira na to koliko je intenzivno i jako svjetlo. Kada promijenite dimer unutar ovih 5 sekundi, svjetlo LED-a se neće promijeniti.



Slika 23 – Analogni nasuprot digitalnog signala

### Sažetak

U stvarnom svijetu, analogni signali se najčešće koriste. Međutim, digitalni signali i obrada preporučuju se kada je u pitanju robotika. Razlozi su: 1) digitalna obrada je jeftinija i pruža veću fleksibilnost u usporedbi s analognom obradom; 2) programi rade u binarnom obliku sličnom digitalnim signalima; i 3) vjerojatnije je da će na analogne signale utjecati šum električnog kruga.

Zbog ovih razloga, analogni signali se često pretvaraju u digitalne signale. Treba napomenuti da se svi analogni signali, i ulazi i izlazi, mogu pretvoriti u digitalne, ali ne i obrnuto.

## 2. Osnove programiranja s Arduino IDE

⇒ **Broj sudionika:** 1-10 po voditelju

⇒ **Trajanje:** 3h

- ⇒ **Metode podučavanja:** prezentacija, vođene instrukcije, iskustveno učenje
- ⇒ **Potrebni materijali:** prezentacija, Arduino IDE i ploče, računalo (1 po sudioniku) i stabilna internetska veza

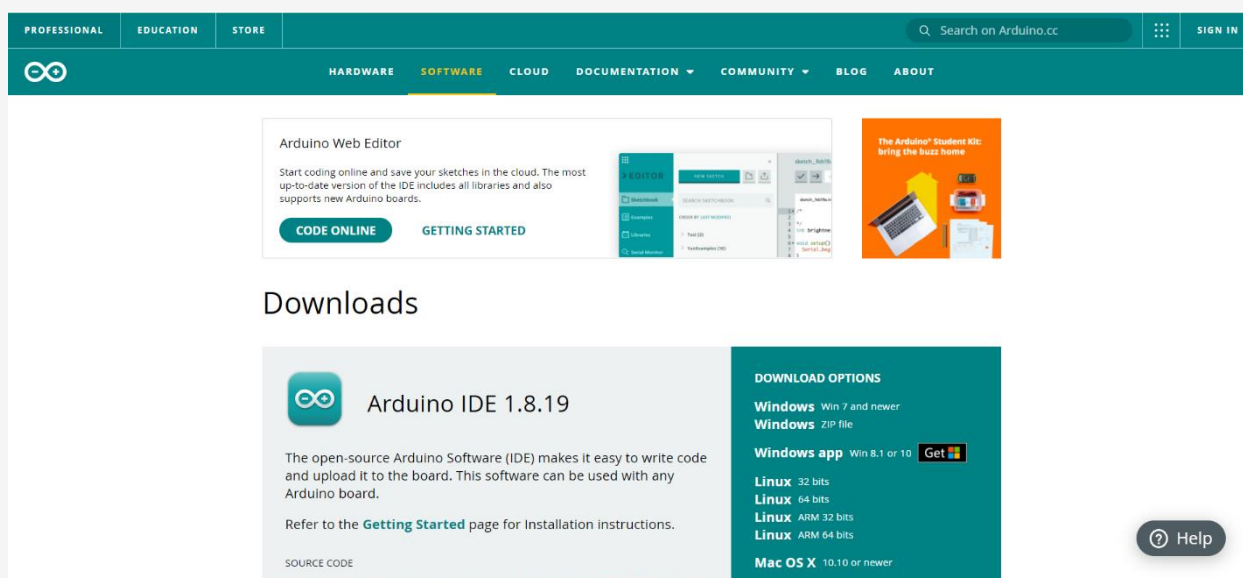
## 2.1. Postavljanje Arduino IDE i osnovnih naredbi

U ovom ćemo odjeljku naučiti kako preuzeti i pokrenuti Arduino IDE po prvi put. Također ćemo proći kroz neke osnovne funkcije Arduino IDE kako bismo se upoznali s programom.

### Kako preuzeti Arduino IDE:

U nastavku slijede upute za preuzimanje i postavljanje Arduino IDE:

1. Idite na <https://www.arduino.cc/>
2. Kliknite na Software. Automatski će se otvoriti stranica prikazana ispod na slici 7.

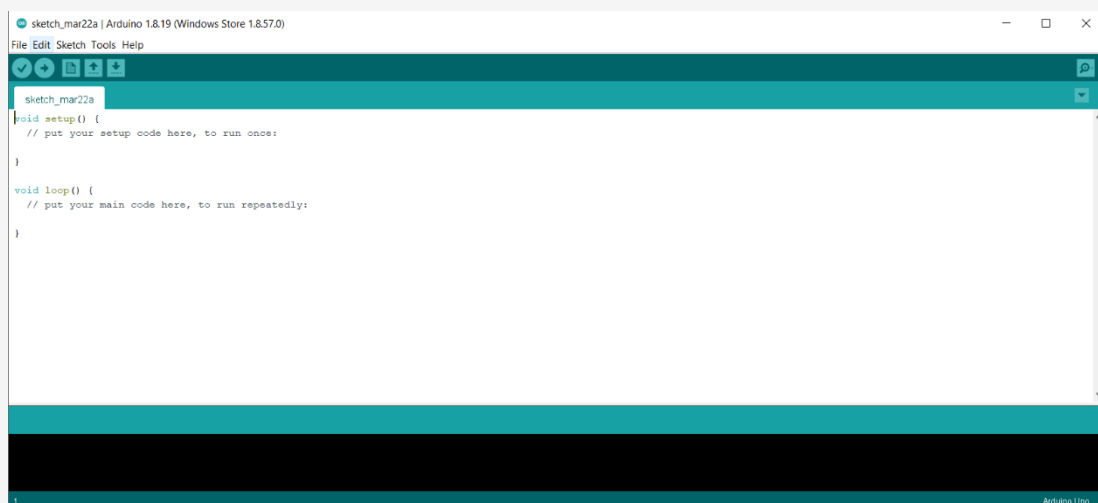


Slika 24 – Preuzimanje Arduino IDE

3. Kliknite na opciju preuzimanja koja odgovara vašem operacijskom sustavu (vidi sliku 7). Ako imate sumnji, kliknite na “Početak rada” kako biste pročitali više informacija o instaliranju softvera za vaše računalo.

Arduino IDE se koristi za kreiranje, otvaranje i promjenu skica. Ploča je definirana skicama koje ispisujemo na računalo kroz Arduino IDE. Možete koristiti gumbе prikazane na vrhu IDE-a ili stavke izbornika.

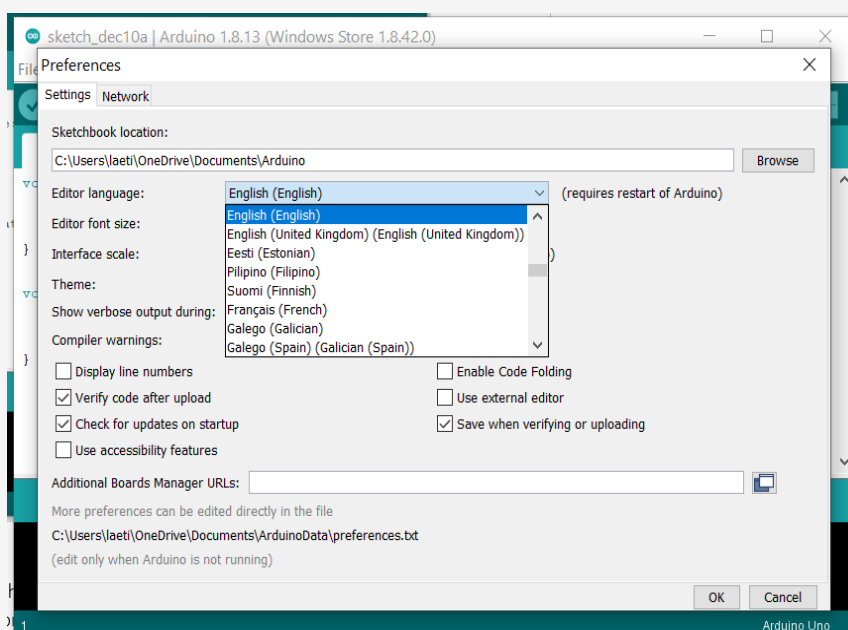
Kada se preuzimanje završi, dolje prikazana stranica će se automatski otvoriti:



Slika 25 – Otvaranje Arduino IDE po prvi put

### Kako promijeniti jezik:

1. Kliknite na DATOTEKA i odaberite PREFERENCE.
2. Pored jezika uređivača prikazuje se padajući izbornik trenutno podržanih jezika.
3. Odaberite željeni jezik s izbornika.
4. Ponovno pokrenite softver da biste koristili odabrani jezik.



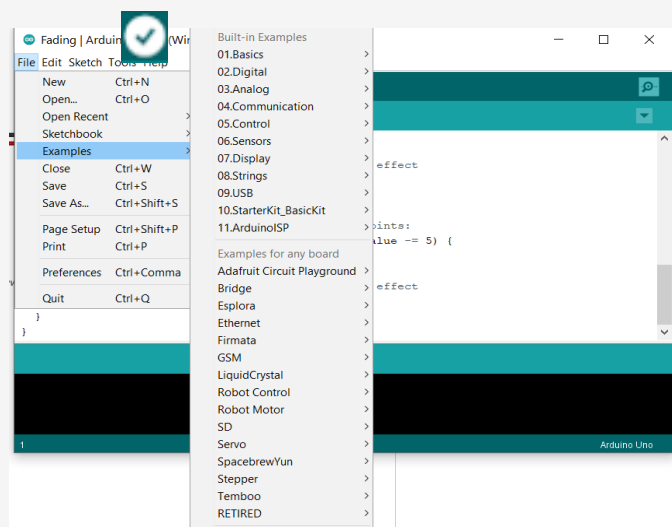
Slika 26 – Izmjena postavki jezika u Arduino IDE

## Kako napraviti novi projekt:

Datoteka -> Novo  
Također je moguće koristiti već stvorene primjere kako biste dobili inspiraciju ili ih reproducirali:

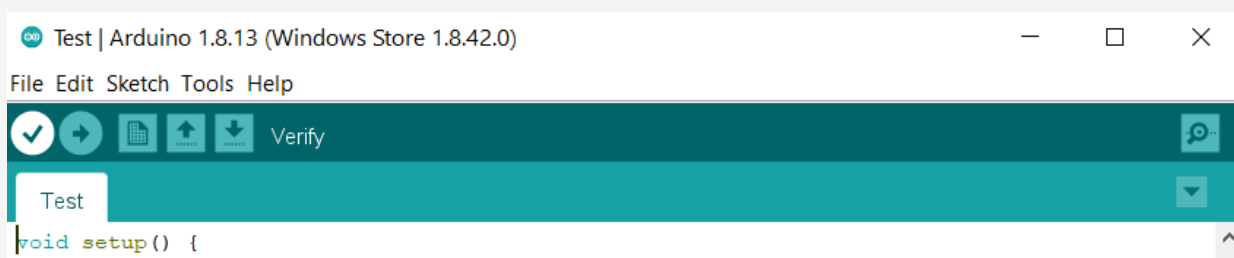
Datoteka -> Primjeri

**Slika 27** – Stvaranje novog projekta u Arduino IDE



## Kako verificirati projekt:

Kliknite lijevo ispod kartice DATOTEKA. Postat će narančasta dok prikuplja informacije. Pričekajte dok se ne vrati u početnu boju.

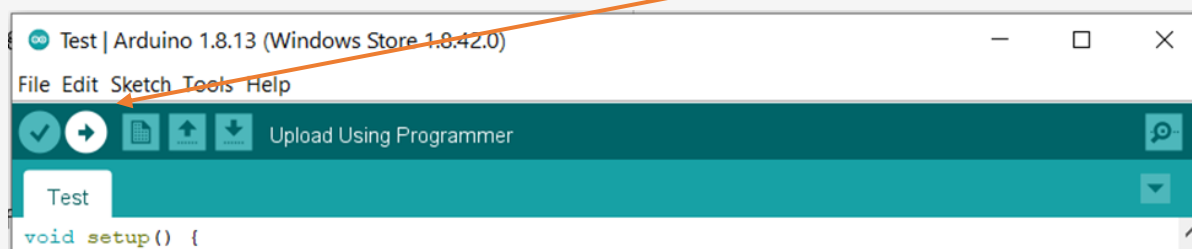


**Slika 28** – Provjera projekta u Arduino IDE



### Kako učitati program:

Arduino ploča treba biti priključena na vaše računalo pomoću USB kabela za prijenos programa. Da biste učitali program, trebete kliknuti na vodoravnu strelicu:



Slika 29 – Prijenos programa u Arduino IDE

Kada se program učitava, strelica će se vratiti na svoju početnu boju.

Ako ste priključili i postavili svoju Arduino ploču prema onome što ste programirali, moći ćete promatrati svoj program u akciji.

## 2.2. Programiranje u Arduino i učitavanje programa na ploču

### Kako programirati u Arduino:

Nakon razumijevanja hardvera Arduino UNO ploče i preuzimanja Arduino softvera, spremni smo za početak programiranja.

Arduino programi se mogu podijeliti u tri glavna dijela:

1. Struktura,
2. Vrijednosti (varijable i konstante) i
3. Funkcije.

U ovom poglavlju naučit ćemo o softverskom programu Arduino, korak po korak, i kako možemo napisati program bez ikakve sintaktičke ili kompilacijske greške.

### Arduino – Programska struktura

Započinjemo sa strukturom. Struktura softwera se sastoji od dvije osnovne funkcije:

Funkcija Setup( ), (postavljanje)

## Funkcija Loop() ,(petlja)

```

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

Slika 30 - Arduino – struktura programa

Funkcija setup() se poziva kada skica započne. Koristimo ga za inicijalizaciju varijabli, načina rada pin-a, početak korištenja knjižnica itd. Funkcija postavljanja će se pokrenuti samo jednom nakon svakog uključivanja ili resetiranja Arduino ploče. Nakon kreiranja funkcije setup() koja inicijalizira i postavlja početne vrijednosti, funkcija loop() radi upravo ono što joj ime sugerira I uzastopno petlja, dopuštajući vašem programu da se promijeni i odgovori. Koristite ga za aktivnu kontrolu Arduino ploče.

## Vrste podataka

Arduino okruženje slično je C++ s podrškom za knjižnicu i ugrađenim pretpostavkama o ciljnom okruženju kako bi se pojednostavio proces kodiranja. Ispod je popis nekih vrsta podataka koji se obično koriste u Arduino:

- boolean (8 bita): točno/netočno
- bajt (8 bita): neoznačeni broj od 0-255
- char (8 bita): označeni broj od -128 do 127.
- riječ (16 bita): neoznačeni broj od 0-65535
- int (16 bita): označeni broj od -32768 do 32767.
- neoznačena duljina (32 bita) - neoznačeni broj od 0-4,294,967,295.

## Arduino – varijable

Varijable u programskom jeziku C, koji koristi Arduino, imaju svojstvo zvano opseg. Opseg je regija programa, a postoje tri mjesta na kojima se varijable mogu deklarirati. Oni su:

- Unutar funkcije ili bloka, koji se nazivaju lokalne varijable.
- U definiciji parametara funkcije, koji se nazivaju formalni parametri.
- Izvan svih funkcija, koje se nazivaju globalne varijable.

Primjer varijabli:

Ovaj primjer stvara cijeli broj nazvan 'countUp', koji je u početku postavljen kao broj nula. Varijabla se povećava za jedan u svakoj petlji.

```
int countUp = 0;           //creates a variable integer called 'countUp'

void setup() {
  Serial.begin(9600);     // use the serial port to print the number
}

void loop() {
  countUp++;             //Adds 1 to the countUp int on every loop
  Serial.println(countUp); // prints out the current state of countUp
  delay(1000);
}
```

**Slika 31** – Primjer Arduino varijabli

Izvor: <https://www.arduino.cc/reference/en/language/variables/data-types/int/>

Operator je simbol koji poručuje prevodiocu da izvrši određene matematičke ili logičke funkcije. Jezik C bogat je ugrađenim operatorima i nudi sljedeće vrste operatora:

- Aritmetički operatori
- Operatori za usporedbu
- Booleovi operatori
- Bitwise operatori
- složeni operateri

Više pojedinosti o svakoj vrsti operatora možete pronaći ovdje:

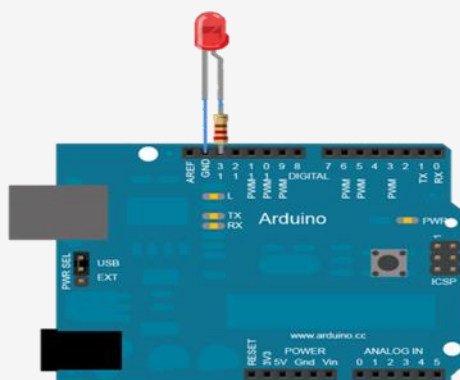
[https://www.tutorialspoint.com/arduino/arduino\\_operators.htm](https://www.tutorialspoint.com/arduino/arduino_operators.htm)

### 2.3. Trepćući Led u Arduino

U ovom dijelu stvaramo i učitavamo jednostavnu skicu koja će uzrokovati da LED dioda više puta treperi paljenjem i isključivanjem u intervalima od 1 sekunde.

Koraci:

- Povežite Arduino na računalo pomoću USB kabela.
- Otvorite IDE
- Odaberite *Tools4Serial* Port. Provjerite je li USB priključak odabran i je li Arduino ploča ispravno spojena.
- Spojite LED na Arduino digitalni pin 13 (kao što je prikazano na donjoj slici). Digitalni pin može detektirati električni signal ili generirati jednu naredbu. U ovom malom projektu ćemo generirati električni signal koji će upaliti LED.



**Slika 32** – Arduino Digital Pin na primjeru trepereće LED diode  
Izvor: <https://www.instructables.com/How-to-Blink-LED-Using-Arduino/>

- Unesite sljedeće u svoju skicu između zagrada { and }, ispod void setup():  
`pinMode(13, IZLAZ); // postaviti digitalni pin 13 na izlaz`

Broj 13 na popisu predstavlja digitalni pin kojem se obraćate. Ovaj pin postavljate na OUTPUT, koji će generirati (izvesti) električni signal. Ako želite da otkrije dolazni električni signal, umjesto toga upotrijebite INPUT. Primijetite da funkcija `pinMode()` završava točkom-zarezom (;). Svaka funkcija u vašim Arduino skicama završit će točkom i zarezom.

- Ponovno spremite svoju skicu kako biste osigurali da odabirom nećete izgubiti ništa od svog rada

Datoteka > Spremi kao.

- Unesite kratki naziv za svoju skicu, a zatim kliknite U redu.

Zapamtite da je naš cilj da LED dioda više puta treperi. Napraviti ćemo funkciju petlje koja će reći Arduino da izvršava uputu uzastopno sve dok se napajanje ne isključi ili netko ne pritisne tipku RESET.

- Unesite kôd prikazan podebljanim slovima nakon odjeljka void setup() na sljedećem popisu kako biste stvorili praznu funkciju petlje.
- Završite ovaj novi odjeljak drugom vitičastom zagradom (}), a zatim ponovno spremite svoju skicu.

```

void setup()
{
  pinMode(13, IZLAZ); // postaviti digitalni pin 13 na izlaz
}
void loop()
{
  // postavite svoj glavni kôd petlje ovdje:
}

```

- Zatim unesite stvarne funkcije u void loop() Arduino na izvršenje.
- Unesite sljedeće između zagrada funkcije petlje, a zatim kliknite *Verify (Provjeri)* kako biste bili sigurni da ste sve ispravno unijeli:

```

digitalWrite(13, HIGH); // uključite digitalni pin 13
delay (1000); // stanka na jednu sekundu
digitalWrite(13, LOW); // isključiti digitalni pin 13
delay (1000); // stanka na jednu sekundu

```

**Funkcija digitalWrite()** kontrolira napon koji izlazi iz digitalnog pina: u ovom slučaju, pin 13 na LED. Postavljanjem drugog parametra ove funkcije na **HIGH**, izlazi "visoki" digitalni napon; tada će struja teći iz pina, a LED će se upaliti.

Kada je LED dioda uključena, svjetlo se zaustavlja na 1 sekundu s odgodom (1000). Funkcija **delay()** uzrokuje da skica ne radi ništa tijekom određenog vremenskog razdoblja—u ovom slučaju, 1000 milisekundi ili 1 sekundu.

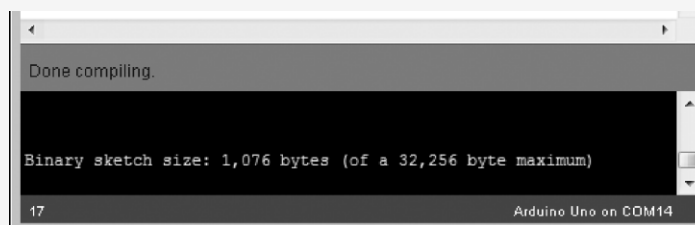
- Zatim isključujemo napon na LED-u pomoću digitalWrite(13, LOW);
  - Konačno, ponovno paziramo 1 sekundu dok je LED dioda isključena, s delay (1000);
- Dovršena skica bi trebala izgledati ovako:

```

void setup()
{
  pinMode(13, IZLAZ); // postaviti digitalni pin 13 na izlaz
}
void loop()
{
  digitalWrite(13, HIGH); // uključite digitalni pin 13
  delay (1000); // stanka na jednu sekundu
  digitalWrite(13, LOW); // isključiti digitalni pin 13
  delay (1000); // stanka na jednu sekundu
}

```

- Spremite svoju skicu.
- Provjerite svoju skicu kako biste bili sigurni da je ispravno napisana kako bi Arduino mogao razumjeti.
- Nakon što je skica provjerena, u prozoru poruke bi se trebala pojaviti napomena, kao što je prikazano u nastavku:



**Slika 33** – Provjera skice u primjeru trepereće LED diode

- Provjerite je li vaša Arduino ploča povezana i kliknite Upload u IDE-u. IDE može ponovno provjeriti vašu skicu i prenijeti je na vašu Arduino ploču.

TX/RX LED diode na vašoj ploči trebale bi treptati tijekom ovog procesa, što znači da vaš program ispravno funkcionira.

### 3. Aplikacije

- ⇒ **Broj sudionika:** 1-10 po voditelju
- ⇒ **Trajanja:** 5.5 sati
- ⇒ **Metode podučavanja:** prezentacija, vođene instrukcija, iskustveno učenje
- ⇒ **Potrebni materijali:** prezentacija, Arduino IDE i ploče, računalo (1 po sudioniku), stabilna internetska veza i relevantni materijali, ovisno o projektu Stable (npr., paintbot)

#### 3.1. Što je robotika?

Robotika je polje u kojem se prožimaju znanost, inženjerstvo i tehnologija. Cilj je proizvesti strojeve, zvane roboti, koji su sposobni zamijeniti, replicirati ili pomoći ljudskim radnjama. Prvotno su roboti napravljeni za obavljanje monotonih zadataka, posebno u industriji. Ali od svog nastanka, proširili su se izvan svoje početne upotrebe. Danas možemo pronaći ogroman izbor robota koji mogu obavljati širok raspon zadataka poput gašenja požara, čišćenja domova i pomoći liječnicima u nevjerojatno zahtjevnim operacijama. Svaki robot uključuje različitu razinu autonomije, počevši od robota pod kontrolom ljudi koji obavljaju zadatke nad kojima

čovjek ima potpunu kontrolu do potpuno autonomnih botova koji obavljaju zadatke bez vanjskih utjecaja.

Svijet robotike se očito širi, ali ipak, možemo pronaći dosljedne karakteristike:

1. "Roboti se sastoje od neke vrste mehaničke konstrukcije. Mehanički aspekt robota pomaže mu u dovršavanju zadataka u okruženju za koje je dizajniran" (Built In, 2022, §3).
2. "Roboti trebaju električne komponente koje kontroliraju i napajaju strojeve" (Built In, 2022, §3).
3. "Roboti sadrže barem neku razinu računalnog programiranja. Bez skupa kodova koji mu govori što da radi, robot bi bio samo još jedan komad jednostavne mašinerije. Umetanje programa u robota daje mu mogućnost da zna kada i kako izvršiti zadatak" (Built In, 2022, §3).

### 3.2 Vrste robota

U današnjem svijetu možemo pronaći niz robota koji se koriste za različite primjene. Kako tehnologija napreduje, roboti postaju sve važniji i igraju važnu ulogu u našem svakodnevnom životu. Postoji mnogo vrsta robota i oni se jako razlikuju od jednog do drugog, od mini robota do ogromnih industrijskih robota, od uslužnih robota do robota za zabavu.

Robote možemo klasificirati na mnogo različitih načina, s obzirom na njihove vrste i primjenu. Svaki robot ima svoje jedinstvene značajke i može se jako razlikovati po veličini, obliku i mogućnostima. Ipak, mnogi roboti dijele različite značajke. Slijedi 13 kategorija koje možemo koristiti za klasifikaciju robota.

#### Industrijski roboti

U industriji robota možemo pronaći uglavnom šest vrsta robota: zglobni roboti, kartezijanski roboti, SCARA roboti, cilindrični roboti, delta roboti i polarni roboti.

- Zglobni robot: po svojoj mehaničkoj konfiguraciji nalikuje ljudskoj ruci. Ruka je spojena na podnožje spojem za uvijanje. Broj rotacijskih zglobova koji povezuju karike u kraku može se kretati od dva zgloba do deset zglobova, a svaki zglob pruža dodatni stupanj slobode (Analytics Insight, 2021.).

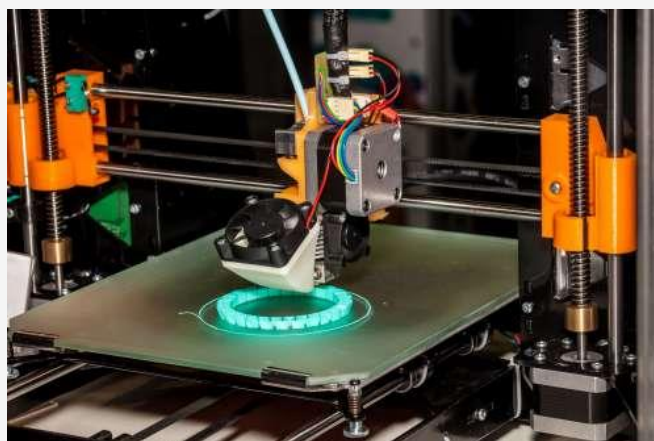




**Slika 34** – Zglobni robot

Izvor: <https://diy-robotics.com/article/articulated-robots/>

- Kartezijan: također se nazivaju pravolinijski ili portalni roboti, kartezijanski roboti imaju tri linearna zgloba koji koriste kartezijanski koordinatni sustav (X, Y i Z). Također mogu imati pričvršćeno zapešće kako bi se omogućilo rotacijsko kretanje. Tri zgloba u obliku prizme, daju linearno kretanje duž osi (Analytics Insight, 2021.).



**Slika 35** – Kartezijanski robot

Izvor: <https://diy-robotics.com/article/what-you-should-know-about-cartesian-robot/>

- SCARA: selektivni usklađeni montažni/zglobni robot ili ruka (SCARA) se češće koristi za potrebe montaže diljem svijeta zbog svoje jednostavne i nesmetane montaže (Analytics Insight, 2021.).

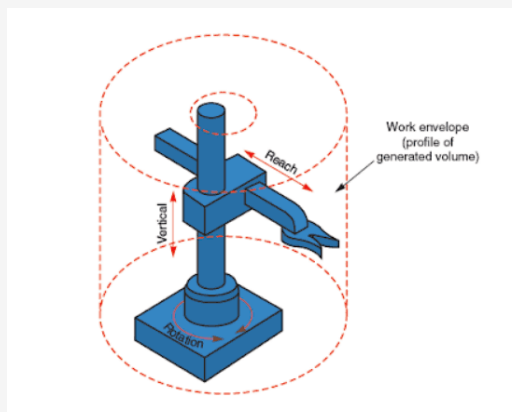




**Slika 36** – SCARA ruka

Izvor: <https://diy-robotics.com/article/scara-robots/>

- Cilindrični: Ovi se roboti općenito koriste za potrebe montaže, točkastog zavarivanja i strojnog tlačnog lijevanja. Imaju minimalno jedan rotacijski spoj na bazi i najmanje jedan prizmatični spoj za spajanje karika. Rotacijski zglob koristi rotacijsko gibanje duž osi zgloba, dok se zglob u obliku prizme giba linearno (Analytics Insight, 2021.).



**Slika 37** – Cilindrična radna omotnica

Izvor: <https://learnmech.com/cylindrical-robot-diagram-construction-applications/>

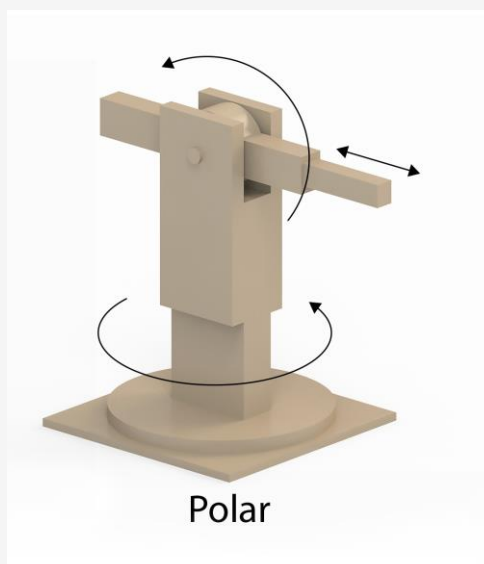
- Delta roboti: također se nazivaju "roboti pauci", koriste tri motora postavljena na bazu za pokretanje kontrolnih ruku koje se nalaze na poziciji zapešća. Osnovni delta roboti su jedinice s 3 osi, ali su dostupni i modeli s 4 i 6 osi (Analytics Insight, 2021.).



**Slika 38**– Delta roboti

Izvor: <https://diy-robotics.com/article/top-six-types-industrial-robots-2020/>

- Polar: Također poznat kao sferni roboti, u ovoj konfiguraciji ruka je povezana s bazom pomoću zavrnog zgloba i kombinacije dva rotirajuća zgloba i jednog linearnog zgloba. Osi tvore polarni koordinatni sustav i stvaraju radni omotač sfernog oblika (Analytics Insight, 2021.).



**Polar**

**Slika 39** – Polarni roboti

Izvor: <https://www.howtorobot.com/expert-insight/industrial-robot-types-and-their-different-uses>

Osim industrijskih robota, postoje neograničene vrste robota u svim područjima kao što su obrazovanje, sigurnost, svemirska znanost, medicina i još mnogo toga. Ovdje možete pronaći neke primjere tih robota, ali zapamtite da ih možete pronaći mnogo više.

### Domaći roboti

Također se nazivaju i potrošački roboti su roboti koje možete kupiti i koristiti samo za zabavu ili da vam pomognu u zadacima i poslovima. Ovi roboti se koriste za obavljanje kućanskih zadataka koji uključuju robote za čišćenje bazena, robotske usisavače, robote za čišćenje

oluka, pomoćnike robota s umjetnom inteligencijom i sve veći izbor robotskih igračaka i kompleta. Ova kategorija može uključivati robote za zabavu, oni su dizajnirani da pobuđuju ljudske emocije kako bi nas zabavili.

### Roboti za vojne potrebe i potrebe odgovora na katastrofe

Roboti se također koriste u vojsci ili za hitnu reakciju kada situacija može biti preopasna za ljude. Robotski dronovi, detektori mina i senzorski uređaji uobičajeni su na bojnopolju, ali zahtijevaju izravnu ljudsku kontrolu. Daljinski upravljana vozila sprječavaju gubitak ljudskih života koji bi se dogodio da su vojnici na terenu umjesto iza kontrolora (Stanford Edu, 2022.). Ipak, upotreba robota i umjetne inteligencije u vojnim operacijama prilično je kontroverzna i može izazvati mnoge legitimne etičke probleme. Nadalje, ove vrste robota su u stanju izdržati ekstremne uvjete i prijeći teška polja. Ovi roboti obavljaju opasne poslove poput traženja preživjelih nakon hitne situacije i pomaganja u drugim ključnim aktivnostima na mjestu katastrofe (Analytics Insight, 2021.).

### Medicinski roboti

Roboti su imali ogroman utjecaj na medicinu. Počeli su prije otprilike 35 godina kada je cilj bio umetnuti sondu u mozak kako bi se dobio biopsijski uzorak. "Danas su medicinski roboti dobro poznati po svojoj ulozi u kirurgiji, točnije korištenju robota, računala i softvera za precizno manipuliranje kirurškim instrumentima kroz jedan ili više malih rezova za različite kirurške zahvate. 3D uvećani prikaz kirurškog polja visoke razlučivosti omogućuje kirurgu rad s velikom preciznošću i kontrolom" (NCBI, 2019.). Jedan od najpoznatijih medicinskih robota je onaj koji je proizveo da Vinci, a odobrila ga je FDA 2000. godine i navodno je korišten za izvođenje preko 6 milijuna operacija diljem svijeta (NCBI, 2019.).

### Uslužni roboti

"Međunarodna organizacija za standardizaciju definira "servisnog robota" kao robota "koji obavlja korisne zadatke za ljude ili opremu, isključujući aplikacije industrijske automatizacije" (IFR, 2022.). Najčešći tipovi uslužnih robota su roboti za pomoć i roboti za ugostiteljstvo. Guardforce Hong Kong, na primjer, lansirao je uslužnog robota koji automatizira registraciju posjetitelja, kontrolira pristupne točke putem prepoznavanja lica i nudi multimedijске informacije, što ga čini savršenim za poslovne zgrade, događaje i izložbe. Ista tvrtka ima ugostiteljskog robota za trgovačke centre koji nudi vodič za kupovinu, funkciju rukovanja i chata te mogućnost preuzimanja kupona (Guardforce, 2022.).

### Koboti



Suradnički roboti ili "koboti" rade zajedno s ljudima u zajedničkom okruženju kako bi obavljali svoje zadatke. Na primjer, robotska ruka Sawyer pomaže radnicima u staklenicima u odabiru biljaka. Mitsubishi robot nudi kavu u kiosku Café X u Hong Kongu.

### Dronovi

Nazivaju se i bespilotne letjelice (UAV), što je zapravo letjelica bez ljudskog pilota. Dronovi mogu biti različitih veličina i različite razine autonomije. Ova vrsta robota napala je širok raspon industrija diljem svijeta i pomaže u obavljanju mnogih aktivnosti kao što su čišćenje atmosfere, snimanje iz zraka i dostava.

### Humanoidni roboti

Ovo je vjerojatno vrsta robota na koju većina ljudi pomisli kada pomisli na robota. Ovi roboti izgledaju kao ljudi i mogu oponašati ljudsko ponašanje. Obično obavljaju aktivnosti slične ljudima (poput trčanja, skakanja i nošenja predmeta) i ponekad su dizajnirani da izgledaju poput nas, čak imaju ljudska lica i izraze (Built In, 2022.).

### Edukativni roboti

Edukativna robotika nova je disciplina osmišljena kako bi učenike upoznala s robotikom i programiranjem na interaktivan način od najranije dobi. Obrazovna robotika studentima pruža sve što im je potrebno za jednostavnu izradu i programiranje robota sposobnog za obavljanje različitih zadataka, a složenost discipline uvijek je prilagođena dobi učenika (Iberdrola, 2022.).

## 3.3 Upravljanje istosmjernim motorom sa štitom motora

Obično roboti i automatizirani uređaji sadrže pokretne dijelove. Kretanje je omogućeno motorima koji su programirani da rade na određeni način. U ovom poglavlju istražujemo programiranje osnovnog istosmjernog motora, motora koji se može vrtjeti u smjeru kazaljke na satu i suprotno. DC motori se općenito koriste u igračkama, kao i u električnim uređajima kao što su mikseri i druge kuhinjske naprave.

Moguće je pokrenuti DC motor s programibilnom pločom (npr. Arduino) koristeći samo nekoliko komponenti. Međutim, radi jednostavnosti, uopćeno je da je najbolje rješenje spojiti istosmjerni motor sa štitom motora.

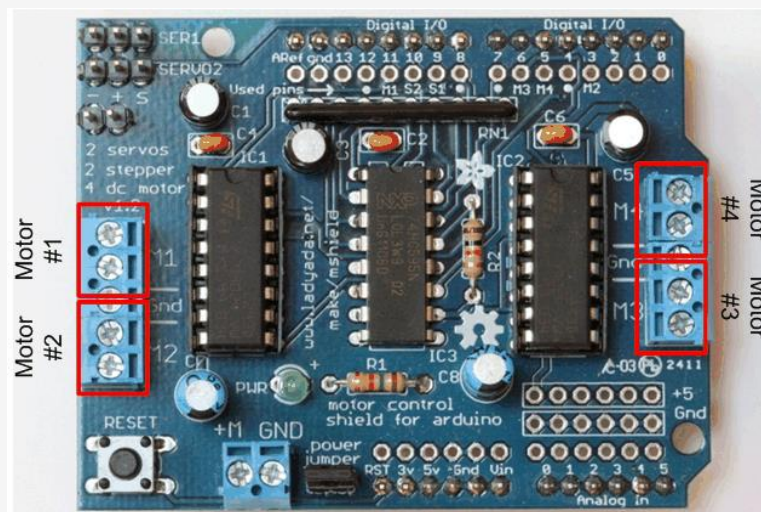
Postoje dvije glavne strategije za povezivanje DC motora na Arduino ploču. Prvi uključuje korištenje Mosfeta i diode, drugi se oslanja na elektroničku komponentu zvanu štit motora.

Štit motora omogućuje nam zaobići niz kompliciranih ožičenja koja su potrebna za rad jednog ili više DC motora.



Postoji nekoliko motornih štitova. Za sljedeću vježbu trebat će vam Adafruit motorni štit V1. Pazite da odaberete verziju 1, a ne verziju 2 Adafruit motornog štita, jer programiranje jednog nije kompatibilno s drugim.

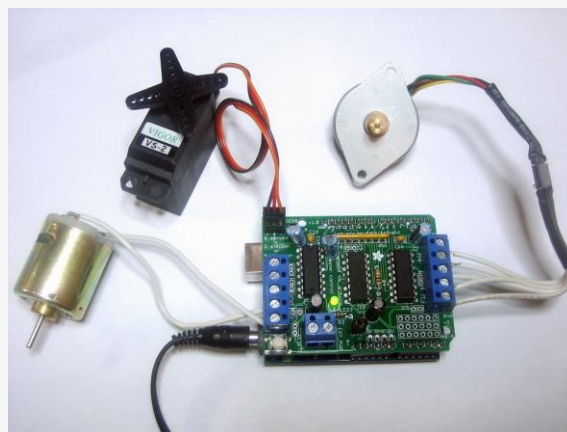
Adafruit motor shield V1 sadrži dva L293D čipa. Iz tog razloga može istovremeno pokretati do 4 istosmjerna motora.



Slika 40 – Adafruit motorni štit V1

Izvor: <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>

Osim DC motora, moguće je pokretati servo motore i koračne motore s adafruit motor shield-om.



Slika 41 – povezan Adafruit motorni štit V1

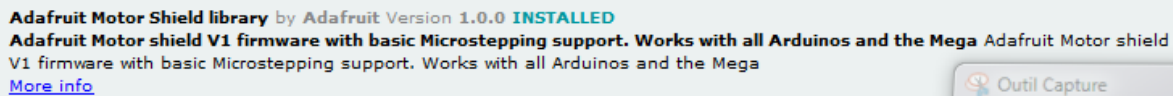
Izvor: <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>

Možete spojiti jedan DC motor na štit spajanjem kabela motora na M1, M2, M3 ili M4.

Kada završite, možete započeti sa zadatkom kodiranja.

Kako bismo programirali istosmjerni motor spojen na naš štit motora, prvo moramo instalirati adafruit biblioteku štita motora. Naći ćete ga u imeniku knjižnice traženjem "AF motor".

Idite na Sketch > Include Library > Manage Libraries...



**Adafruit Motor Shield library** by Adafruit Version 1.0.0 **INSTALLED**  
 Adafruit Motor shield V1 firmware with basic Microstepping support. Works with all Arduinos and the Mega  
 V1 firmware with basic Microstepping support. Works with all Arduinos and the Mega  
[More info](#)

U ovoj fazi možete započeti programiranje štita motora. Jednostavan kôd za okretanje istosmjernog motora je sljedeći:

```
#include <AFMotor.h>
```

```
AF_DCMotor motor1(1); // Na štitu motora definiramo motor pričvršćen na M1
```

```
void setup()
```

```
{
```

```
motor1.setSpeed(100); // Definiramo brzinu kojom će se motor okretati
```

```
}
```

```
void loop()
```

```
{
```

```
motor1.run(BACKWARD); // motor će se okretati u smjeru kazaljke na satu ili u suprotnom  
smjeru, ovisno o načinu na koji ste ga spojili na štiti motora. Da bi se vrtio u suprotnom smjeru,  
trebate zamijeniti NATRAG sa NAPRIJED.
```

```
delay(10000); // vrtit će se 10 sekundi
```

```
motor1.run(RELEASE); // zaustavit će se na 1 sekundu i ponovno početi od vrha funkcije petlje
```

```
delay(1000);
```

```
}
```

### 3.4. Izrada paintbota upotrebom DC motora i Arduina

Moguće je implementirati programiranje DC motora u umjetničkom okruženju kako bi se postiglo nešto interaktivno i kreativno. Paintbot spaja umjetnost s elektronikom i to je stroj koji koristi boju za stvaranje jedinstvenih umjetničkih djela.



**Slika 42** – Umjetnički rad pomoću bota za slikanje

Izvor: <https://girlsinstem.eu/>

PaintBot se sastoji od okretnog stola koji se okreće različitim brzinama. Na okretni stol možete postaviti list papira ili malo platno. Pažljivim ispuštanjem kapi boje iznad rotirajućeg lima, dobivate jedinstveno remek-djelo koje ste stvorili vi i PaintBot.

**Slika 43** – Komponente paintbot-a

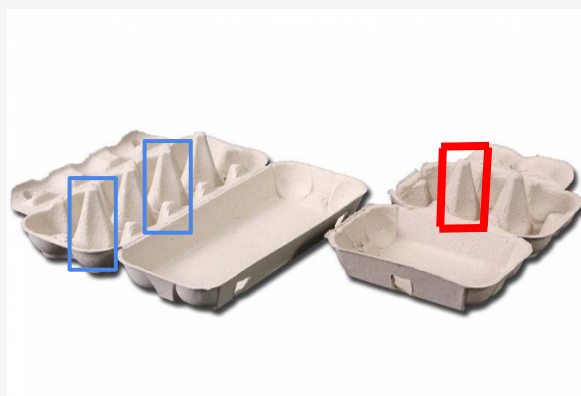
Izvor: <https://girlsinstem.eu/>



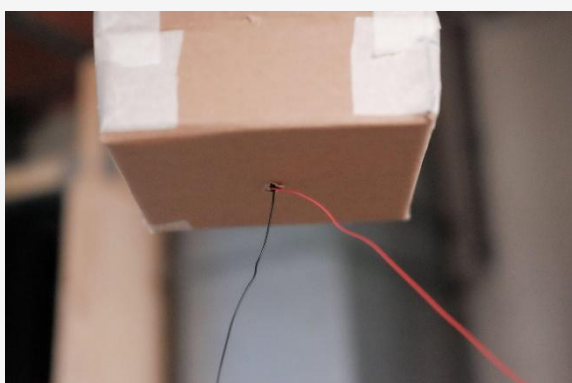
### Izradite kutiju

- A. Uzmite crveni i crni komad žice od najmanje 30 cm svaki i spojite ga na motor. (Pazite da ne izrežete premale komadiće jer to otežava povezivanje motora u PaintBotu s ostatkom elektroničkog kruga. Što je vaša kutija veća, žice moraju biti duže)
- B. Koristite kutiju za jaja kao stalak za motor

1. Skinite poklopac kutije za jaja, ne treba nam.
  2. Trebamo jedan od vrhova kutije za jaja i 4 mjesta za jaja koja ga okružuju. Izrežite između 3. i 4. mjesta jajeta i susjednog vrha.
  3. Odrežite vrh vrha. Bolje je početi rezati mali komad i naknadno ga prilagoditi nego odmah previše rezati.
  4. Postavite motor odozdo prema gore s osovinom motora prema gore i žicama prema dolje. Motor bi trebao čvrsto stati u rupu koju ste upravo izrezali. Možete ga pokušati pažljivo gurnuti ili odrezati malo više s vrha. Važno je da motor čvrsto pristaje i da je konstrukcija robusna jer čini bazu okretnog stola. Za dodatnu stabilnost i potporu možete koristiti traku ili možete staviti čačkalice ispod motora kroz kutiju za jaja.
- C. Napravite malu rupu u sredini kartonske kutije. Provucite žice motora kroz rupu koja ide od unutarnje strane kutije prema vanjskoj ili donjoj strani kutije. Na njega stavite držač kartona za jaja.
- D. Uvjerite se da je držač kartona za jaja postavljen u sredinu kutije i čvrsto ga pričvrstite na kutiju. To je vrlo važno jer mora izdržati silu rotacionog motora.
- E. Stavite kartonski disk (okrugli ili bilo kojeg drugog oblika) na crveni (3D ispisani) spojnik motora. Provjerite je li manji od kutije kako bi se mogao slobodno okretati.



Slika 44 – Korak B paintbota



Slika 45 – Korak C paintbota



Slika 46 – Korak D paintbota



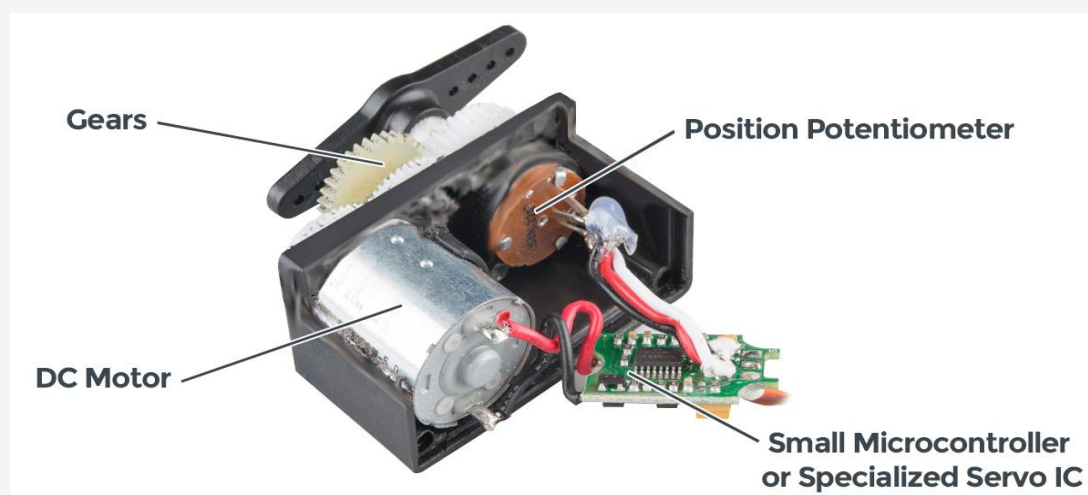
## Spojite elektroniku

Nakon izrade kutije, možete jednostavno spojiti istosmjerni motor koji je pričvršćen na štit motora i programirati ga da promijeni brzinu i smjer kako biste stvorili svoje umjetničko djelo!

### 3.5 Izradite interaktivnu papirnatu igračku sa servo motorom

Servomotor je svaki sustav na motorni pogon s ugrađenim elementom povratne sprege.

Ako otvorite standardni servo motor, gotovo uvijek ćete pronaći tri osnovne komponente: DC motor, upravljački krug i potencijometar ili sličan mehanizam povratne sprege. DC motor je pričvršćen na mjenjač i izlazno/pogonsko vratilo kako bi se povećala brzina i zakretni moment motora. DC motor pokreće izlaznu osovinu. Kontrolni krug tumači signale koje šalje regulator, a potencijometar djeluje kao povratna informacija za krug regulatora za praćenje položaja izlaznog vratila.



Slika 47 – Servo motor

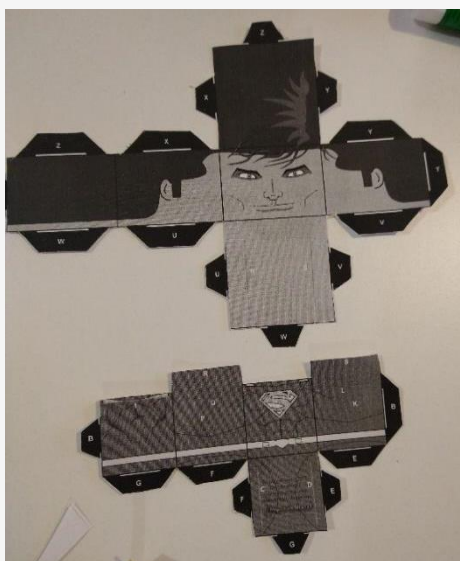
Izvor: <https://www.sparkfun.com/servos>

Servo motori imaju bezbroj primjena. U ovom odjeljku ilustriramo kako upotrijebiti servo motor za izradu interaktivne papirnatu igračku čiju glavu možete okretati lijevo ili desno djelovanjem na potencijometar.

Za ovaj projekt neće biti programiranja, međutim, moguće je postići potpuno iste rezultate spajanjem servo motora s Arduino pločom.

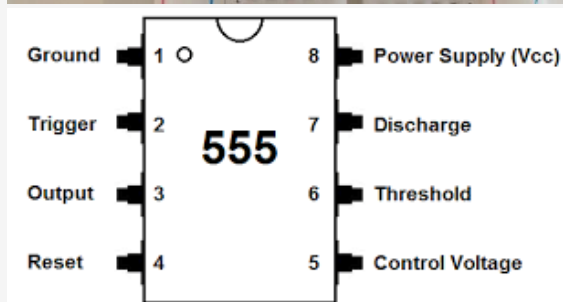
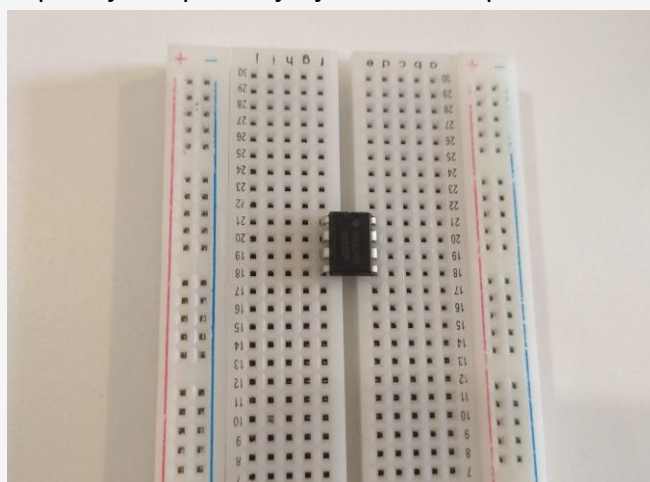
#### Korak 1: Napravite papirnatu igračku

Početak ćemo stvaranjem naše papirnatu igračke. Najprije morate odabrati šablonu, zatim izrezati predložak od papira ili kartona i na kraju sastaviti papirnatu igračku prema uputama. Na [Cubeecraft](#). su dostupni različiti predlošci koje možete izabrati.



## Korak 2: Izgradite elektronički sklop

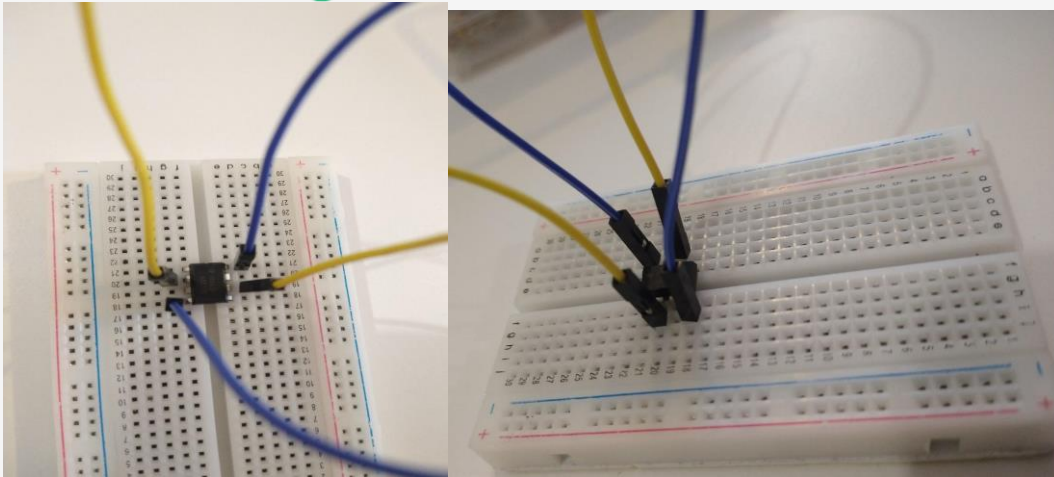
Započnimo postavljanjem NE555 čipa u sredinu matične ploče, baš kao na slici ispod.



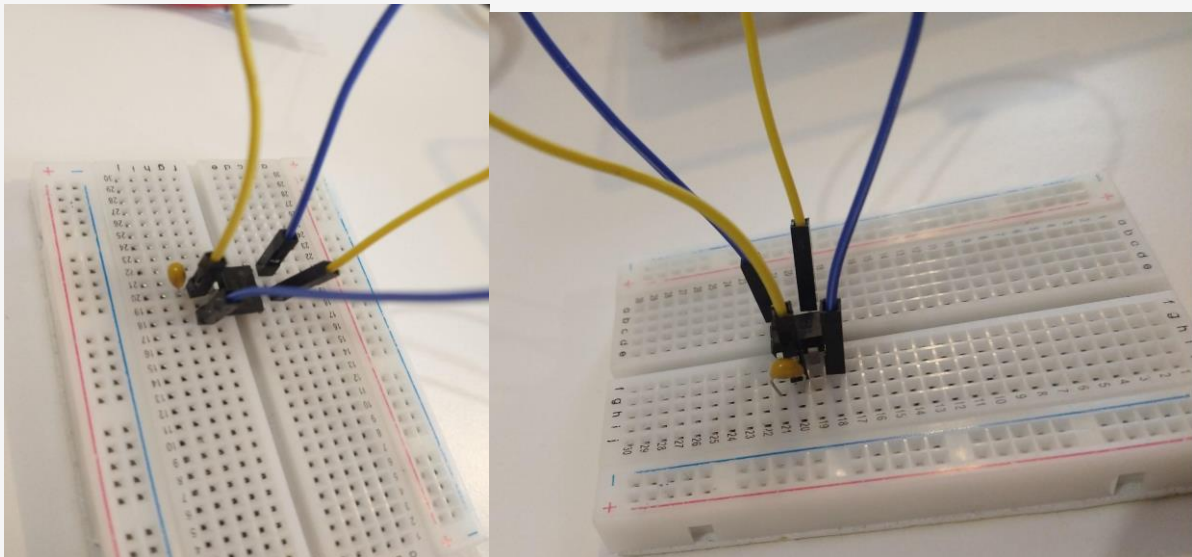
Slika 48 – Timer-pinout

Izvor: <http://www.learningaboutelectronics.com/Articles/555-timer-pinout.php>

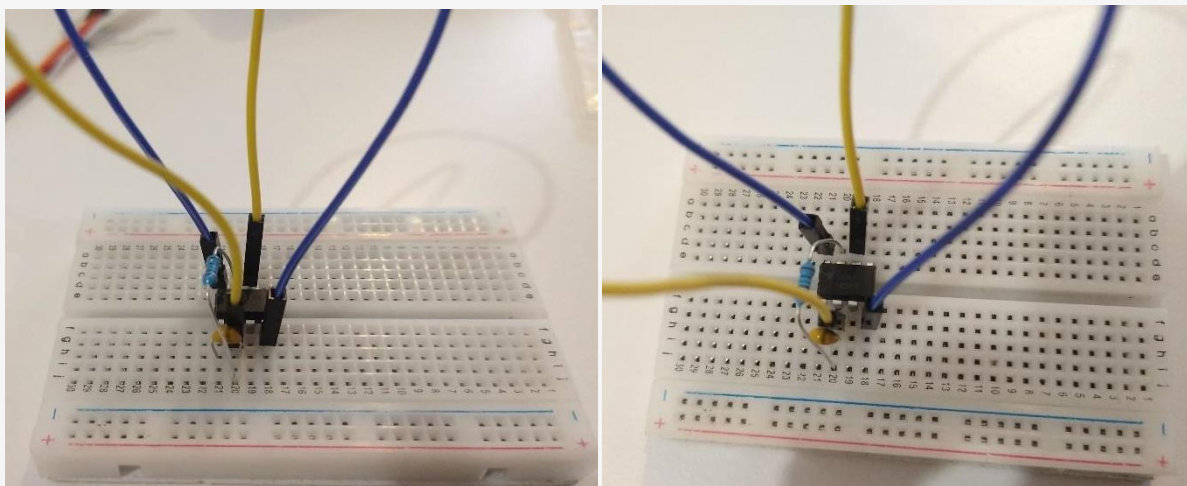
Zatim dodajemo dvije kratkospojne žice, jednu koja povezuje nogu 4 i nogu 8 NE555 i drugu koja povezuje noge 2 i 6.



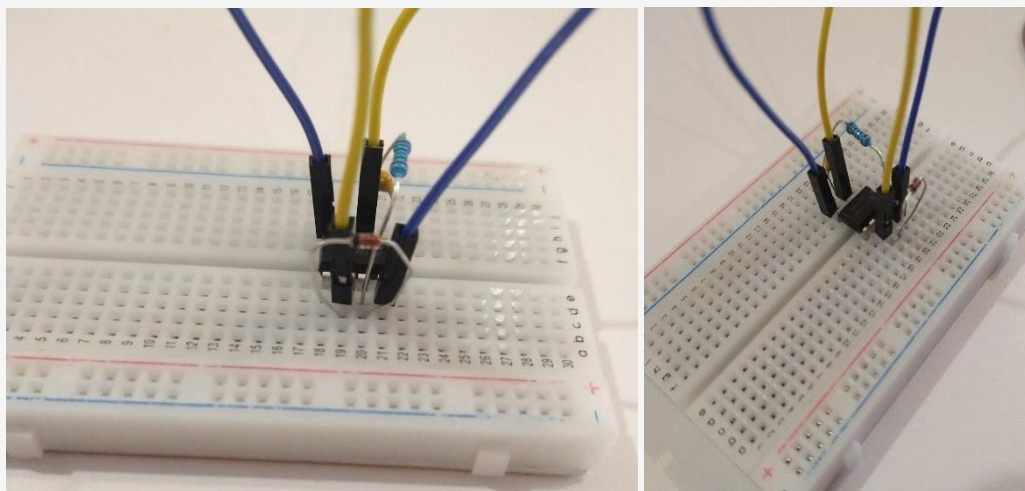
Zatim dodajemo kondenzator koji povezuje noge 1 i 2.



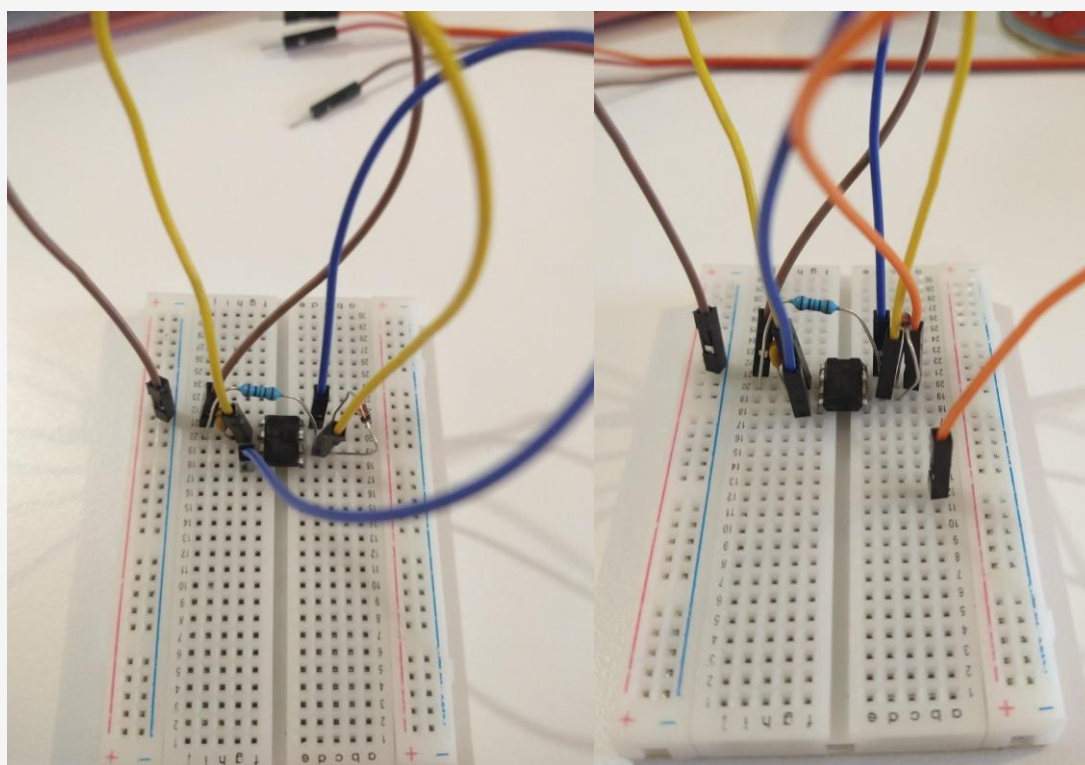
Dodamo otpornik od 220 k ohma koji povezuje noge 2 i 7.

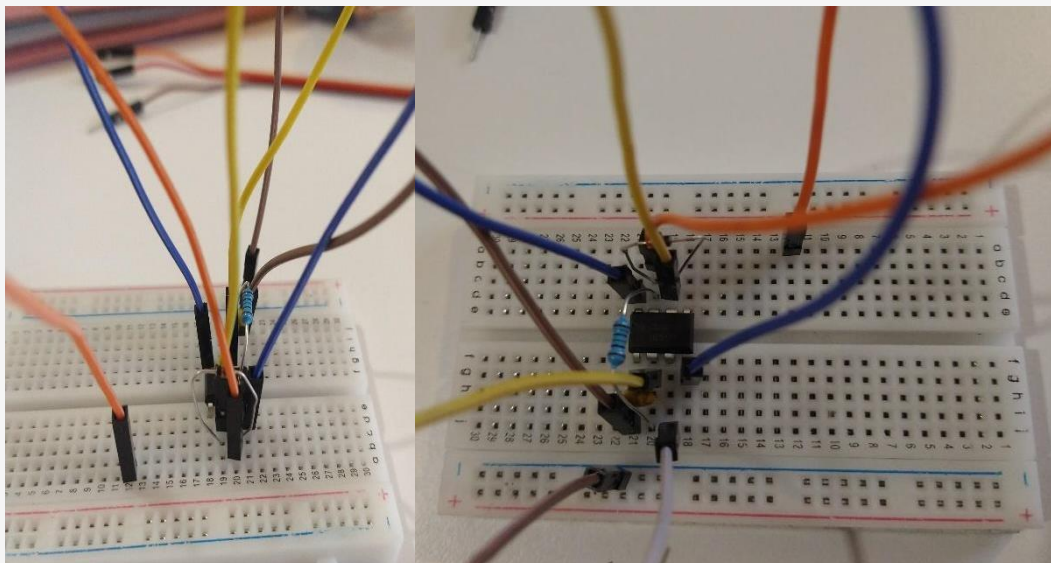


Zatim dodamo Zener diode koje povezuju noge 6 i 7 NE555 kako je prikazano na slici. Pobrinite se da postavite diode u pravom smjeru, tj. Sa crnom prugom na nozi 6.

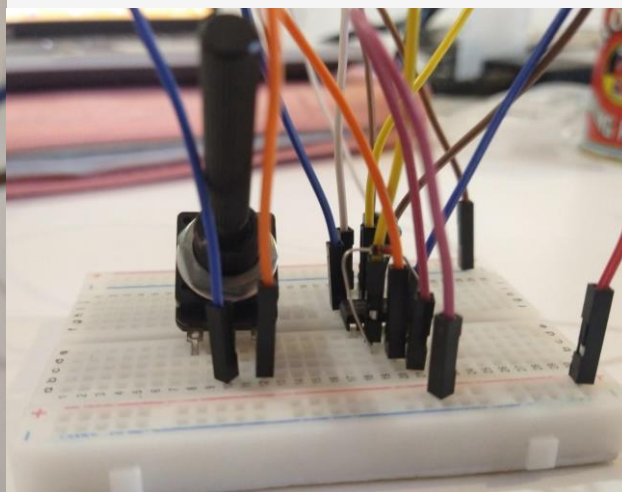
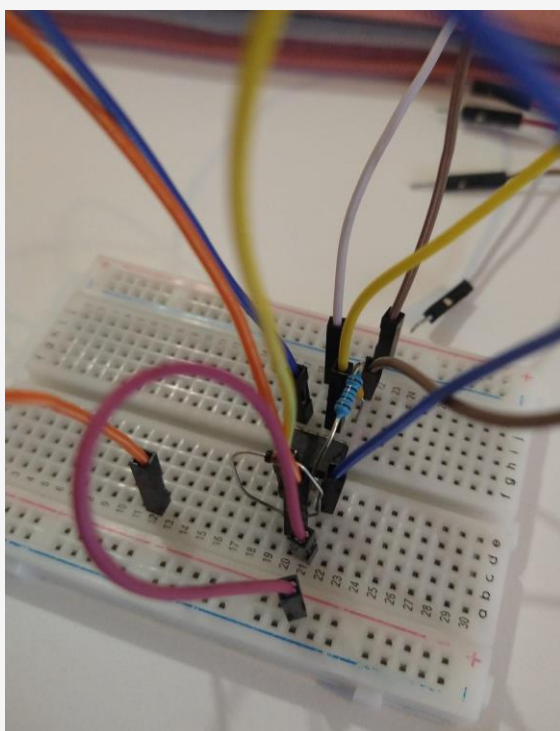


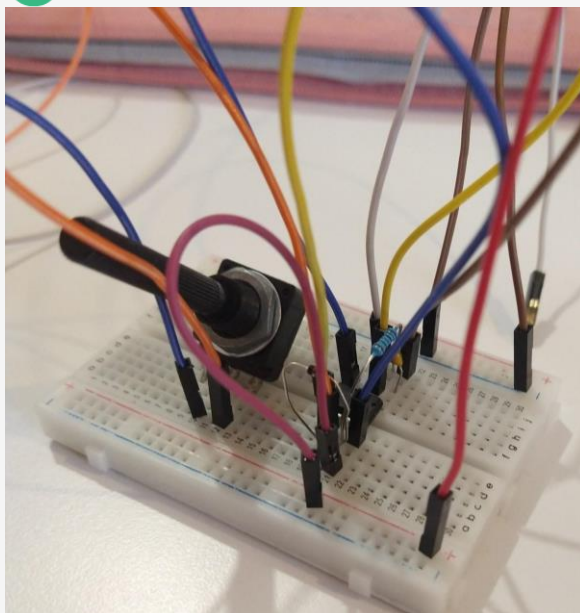
Dodamo kratkospojnu žicu koja ide od noge 1 do negativnog pola, a zatim drugu koja povezuje nogu 7 s drugom linijom dalje u matičnoj ploči. Nova žica će biti spojena s jedne strane na nogu 3 NE555 (za sada ne vodimo računa o drugom kraju žice).



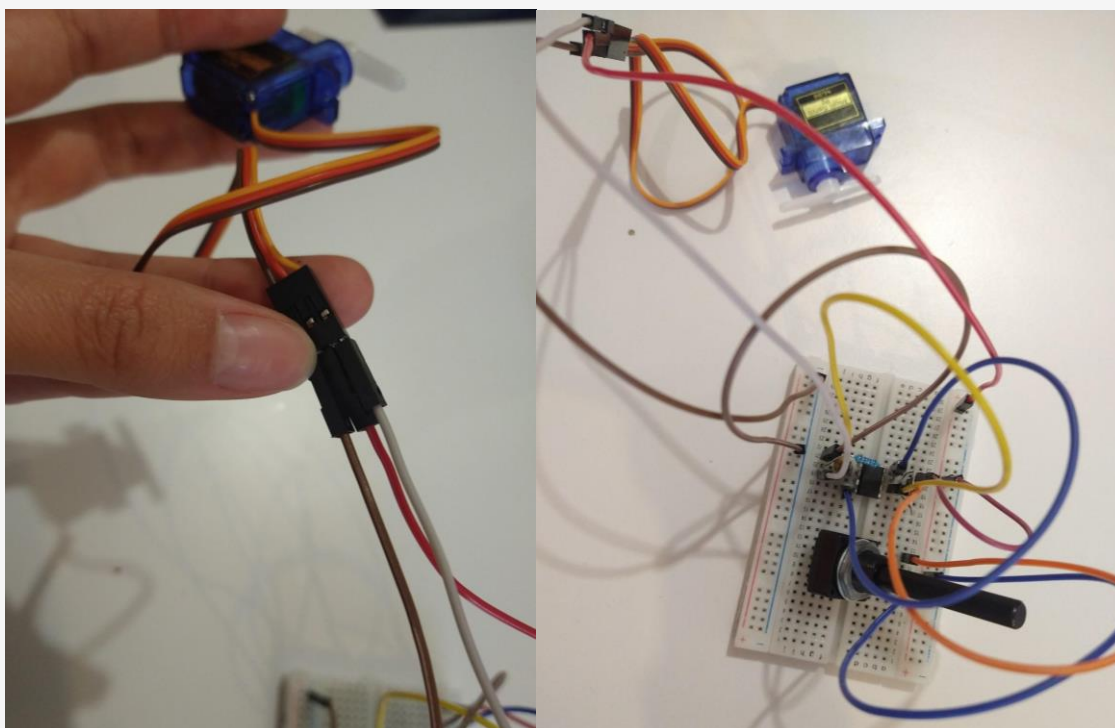


Dodamo kratkospojnu žicu koja spaja nogu 1 na pozitivni pol. Zatim postavljamo potenciometar kao na slici ispod. S nogom na istoj liniji kabela (narančasta) koja je počela od noge 7 NE555. Na liniju gdje se nalazi srednja noga potenciometra dodajemo još jednu žicu koja će se spojiti na pozitivni pol.

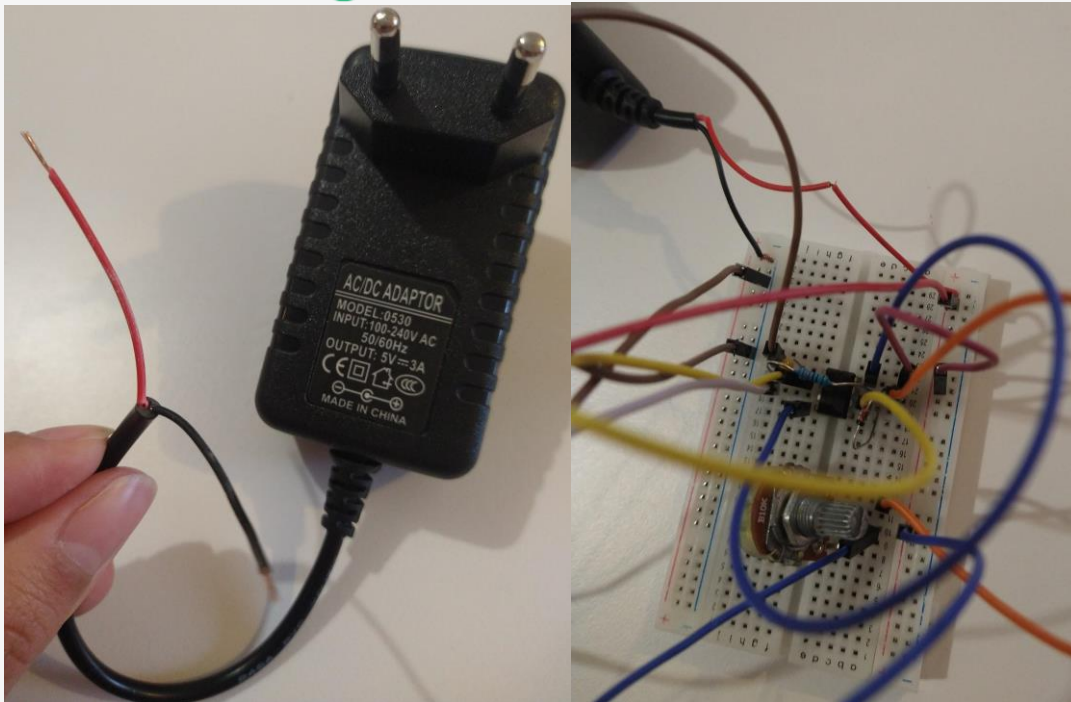




Sada uzimamo (bijelu) žicu kratkospojnika koja je bila spojena na nogu 3 NE555 s jednog kraja i spajamo je na narančasti kabel servo motora. Također, spajamo žicu na pozitivni pol matične ploče s jednog kraja na crvenu žicu serva na drugom kraju. Isto radimo sa žicom koja izlazi iz negativnog pola matične ploče i spaja se na smeđi kabel servo motora (vidi slike ispod).

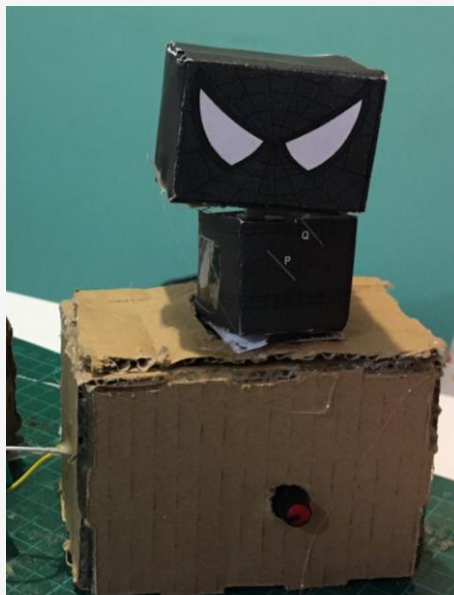


Posljednji korak je da uzmete 5V adapter, izrežete vrh i uklonite plastiku kako bi crveni i crni kabel bili dostupni. Ogolimo malo dva kraja koja ćemo također uvesti u pozitivni, odnosno negativni pol.



To je to, naš krug je gotov!

Da biste dovršili projekt, spojite servo motor na glavu papirnate igračke i sakrijte elektronički krug unutar kartonske kutije. Obavezno izbušite rupu za potenciometar.



### 3.6. Brava vrata na daljinsko upravljanje

IR ili infracrvena komunikacija je uobičajena, jeftina i jednostavna za korištenje bežična komunikacijska tehnologija. IR svjetlo je vrlo slično vidljivom svjetlu, samo što ima nešto veću valnu duljinu. To znači da je IR nevidljiv ljudskom oku - savršen za bežičnu komunikaciju.

Kako biste razumjeli kako IR tehnologija radi, u sljedećem odjeljku izradit ćete bravu za vrata spojenu na Arduino ploču.

## Napravite daljinski upravljaju bravu za vrata koja funkcionira s infracrvenom tehnologijom

U ovom projektu koristit ćemo prethodno prikupljene informacije o infracrvenoj tehnologiji za izradu daljinski upravljane brave za vrata.

Za završetak ovog projekta koristit ćete infracrveni daljinski upravljač, infracrveni prijemnik i servomotor.

### Korak 1: Instalirajte knjižnicu

Idite na [ovu](#) web stranicu i preuzmite biblioteku. Zatim moramo instalirati biblioteku na Arduino IDE softver. Idite na Arduino IDE → Sketches → Uključi biblioteku → IRremote.

### Korak 2: Otkrijte ključ vašeg daljinskog upravljača

Da bismo shvatili kako vaša arduino ploča tumači električne signale koje šalje vaš daljinski upravljač, moramo učitati sljedeći kôd na ploču.

```
/* Pronalaženje kodova ključeva za vaš daljinski upravljač. Više informacija:
https://www.makerguides.com */
```

```
#include <IRremote.h> // uključite biblioteku IRremote
```

```
#define RECEIVER_PIN 13 // definirajte pin IR prijemnika
```

```
IRrecv receiver(RECEIVER_PIN); // stvorite objekt primatelja klase IRrecv
```

```
decode_results results; // kreirajte objekt rezultata klase decode_results
```

```
void setup() {
```

```
  Serial.begin(9600); // započnite serijsku komunikaciju brzinom prijenosa od 9600
```

```
  receiver.enableIRIn(); // omogućite prijemnik
```

```
  receiver.blink13(true); // omogućite treptanje ugrađene LED diode kada se primi IR signal
```

```
}
```

```
void loop() {
```

```
  if (receiver.decode(&results)) { // dekodirajte primljeni signal i pohranite ga u rezultate
```

```
    Serial.println(results.value, HEX); // ispišite vrijednosti u serijskom monitoru
```





```

receiver.resume(); //resetirajte prijemnik za sljedeći kôd
}
}

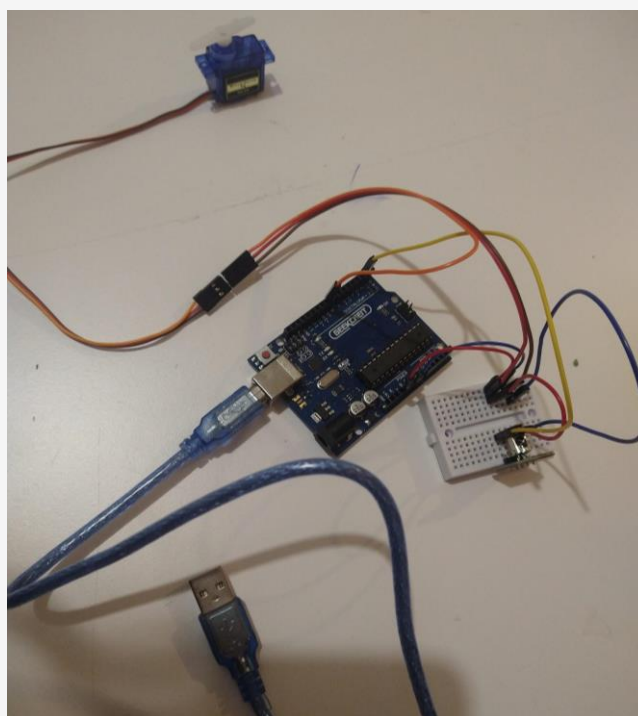
```

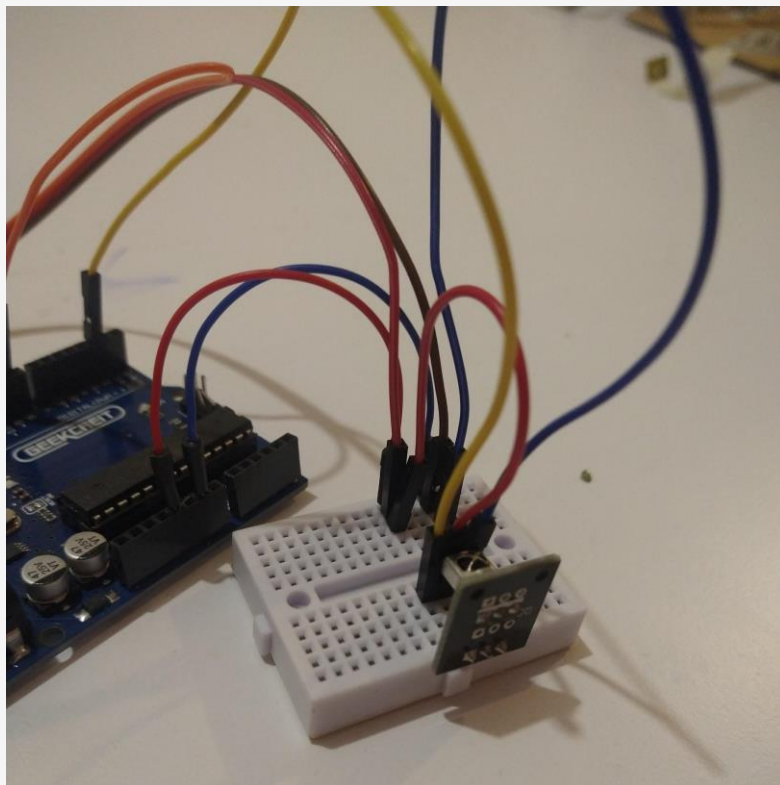
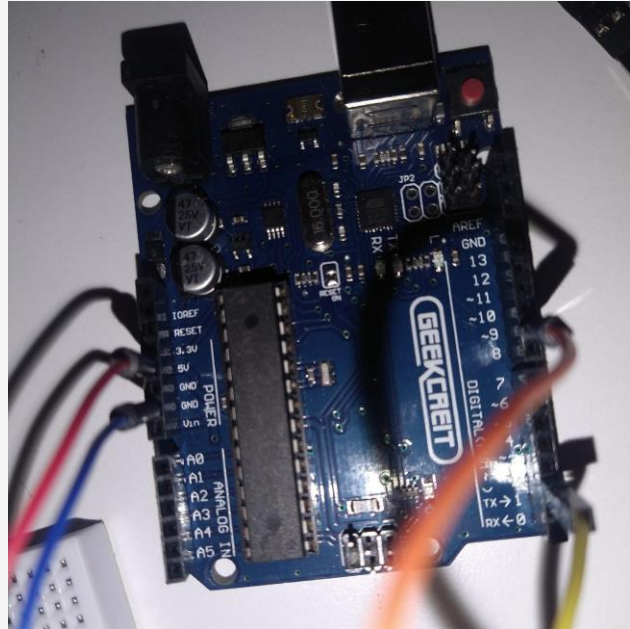
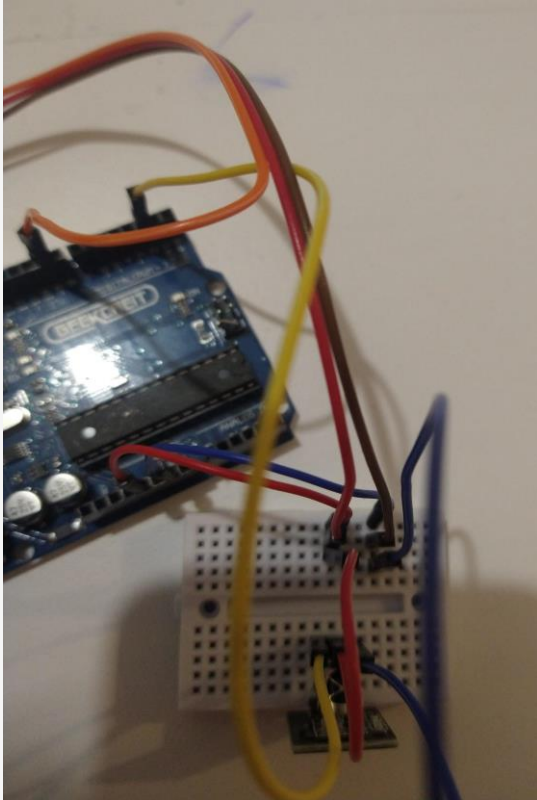
Zatim možete otvoriti Arduino serijski monitor i klikom na gumbе daljinskog upravljača moći ćete vidjeti ključ koji se prikazuje na serijskom monitoru. Svaki gumb vašeg daljinskog upravljača odgovara različitoj tipki. Zabilježite različite ključeve jer će vam ove informacije trebati kasnije.

### Korak 3: Spojite sve komponente

Spojite sve komponente kao što je prikazano na slikama ispod. Budite oprezni s IR prijemom: pozitivni izlaz ide na 5V Arduino ploče, negativni izlaz na uzemljenje i izlaz signala na digitalni pin 2 (pogledajte kôd ispod).

Primijetite da je signalni izlaz servomotora spojen na digitalni pin 9 Arduino ploče.





## Korak 4: Kôd

Trebamo učitati sljedeći kôd na ploču:

```
#include <IRremote.h> //treba kopirati IRremote knjižnicu u arduino biblioteke
#include <Servo.h>
#define up 0xFF906F //gumb za rotaciju u smjeru kazaljke na satu
#define down 0xFFE01F // gumb za rotaciju u smjeru suprotnom od kazaljke na satu

int RECV_PIN = 2; //pin IR prijemnika
Servo servo;
int val; //kut rotacije
bool cwRotation, ccwRotation; //stanja rotacije

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // pokrenite prijarnik
  servo.attach(9); //pin serva
}

void loop()
{
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Primate sljedeću vrijednost

    if (results.value == up)
    {
      cwRotation = !cwRotation; //promijenite vrijednost rotacije
      ccwRotation = false; //ne postoji rotacija u ovom smjeru
    }

    if (results.value == down)
    {
      ccwRotation = !ccwRotation; //promijenite vrijednost rotacije
    }
  }
}
```

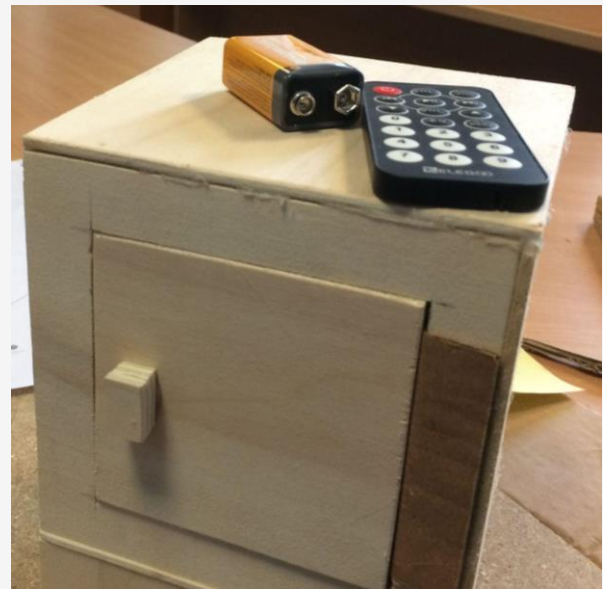
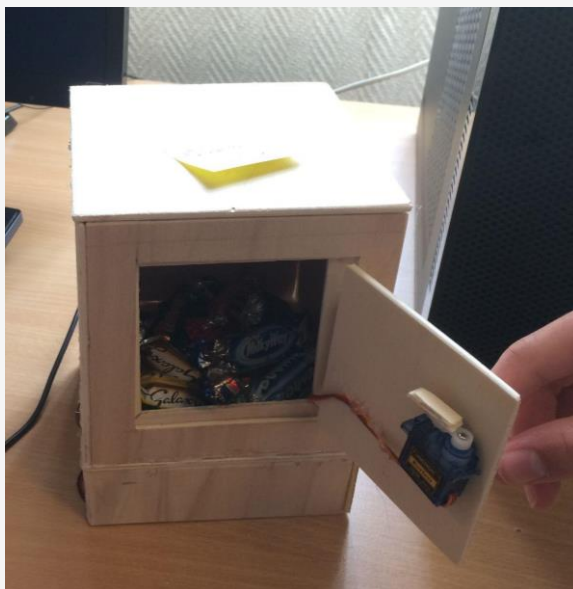
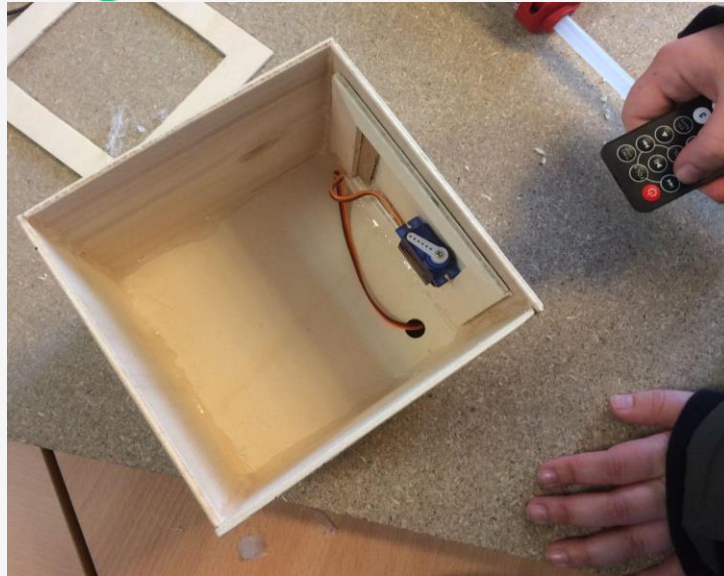


```
    cwRotation = false;    //ne postoji rotacija u ovom smjeru
  }
}
if (cwRotation && (val != 175)) {
  val++;    //za tipku u smjeru kazaljke na satu
}
if (ccwRotation && (val != 0)) {
  val--;    //za tipku u smjeru suprotnom od kazaljke na satu
}
servo.write(val);
delay(20);    //općenita brzina
}
```

### Korak 5: Napravite bravu za vrata

Do sada ste trebali testirati svoj projekt. Ako servo motor reagira na naredbe koje šalje daljinski upravljač, onda je vrijeme da razmislite o zaključavanju vrata. Ovdje možete biti kreativni i zamisliti bilo koju vrstu brave. Predlažemo rješenje koje uključuje pričvršćivanje servo motora izravno na vrata, kao na slici ispod.





### 3.7. Mjerenje temperature, vlažnosti, svjetla i boje

U prethodnim projektima koristili smo motore za postizanje različitih rezultata, od jednostavnog pokretanja motora u smjeru kazaljke na satu ili suprotno od kazaljke na satu do izrade sofisticiranijih naprava poput interaktivne papirnate igračke ili daljinski upravljane brave za vrata.

Sada su motori aktuatori što znači da su namijenjeni za obavljanje određene radnje. Zajedno s aktuatorima, može se odabrati i korištenje nekih senzora. Zapravo smo istražili nekoliko senzora u prethodnom poglavlju. Prvi je bio potenciometar, drugi je bio infracrveni senzor koji prima IRR signale i prevodi ih u računalni kôd za Arduino ploču.

Ovo poglavlje se odnosi na senzore. Cilj je upoznati se s ovom vrstom elektroničkih komponenti kako biste svoje vlastite projekte mogli podići na sljedeću razinu.

Posebno ćemo istraživati senzore koji mjere temperaturu, svjetlost, kao i vlažnost i boju.

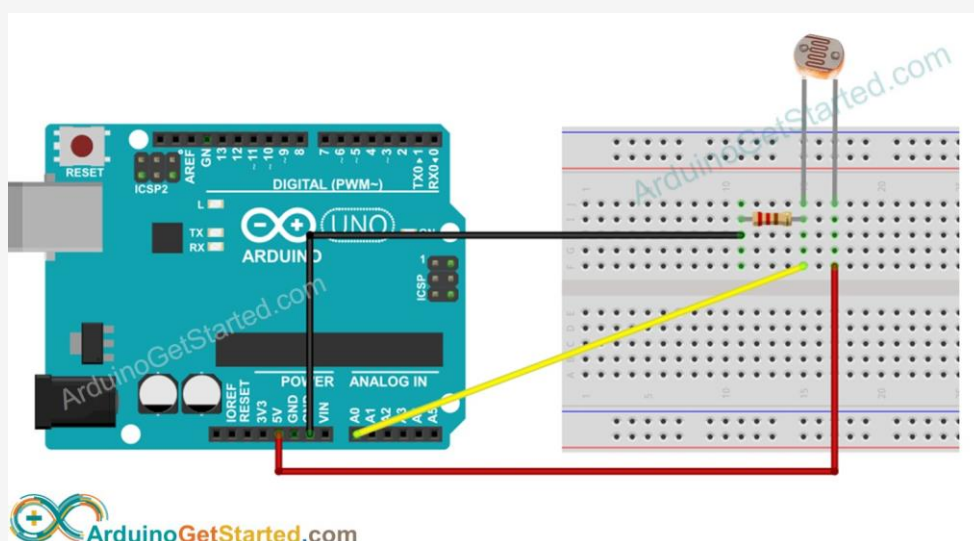
### 3.7.1. Upotreba senzora u Arduinu

Svjetlosni senzor je vrsta otpornika, naziva se otpornik ovisan o svjetlu. Koristi se za detekciju svjetlosti i mjerenje razine svjetline određenog okruženja.

Svjetlosni senzor ima dva pina i budući da je u osnovi otpornik, ne moramo razlikovati ova dva pina.

Što je jačina svjetlosti veća, to je manji otpor koji bilježi svjetlosni senzor. Stoga, mjerenjem otpora svjetlosnog senzora možemo znati koliko je svijetlo okruženje.

Ovako spojite fotootpornik na Arduino ploču.



Slika 49 – Spajanje fotootpornika na Arduino ploču

Izvor: <https://arduinogetstarted.com/tutorials/arduino-light-sensor>

A ovo je jednostavan kôd koji vam omogućuje pregled vrijednosti koje je zabilježio svjetlosni senzor preko serijskog monitora Arduino IDE.

```
void setup() {
  // inicijalizira serijsku komunikaciju na 9600 bita u sekundi:
  Serial.begin(9600);
}

void loop() {
  // čita ulaz na analognom pinu A0 (vrijednost između 0 i 1023)
  int analogValue = analogRead(A0);
  Serial.print("Analog reading: ");
```

```
Serial.print(analogValue); // sirovo analogno čitanje
```

```
delay(500);
```

```
}
```

### 3.7.2. Izradite teremin s Arduinoom i svjetlosnim senzorom

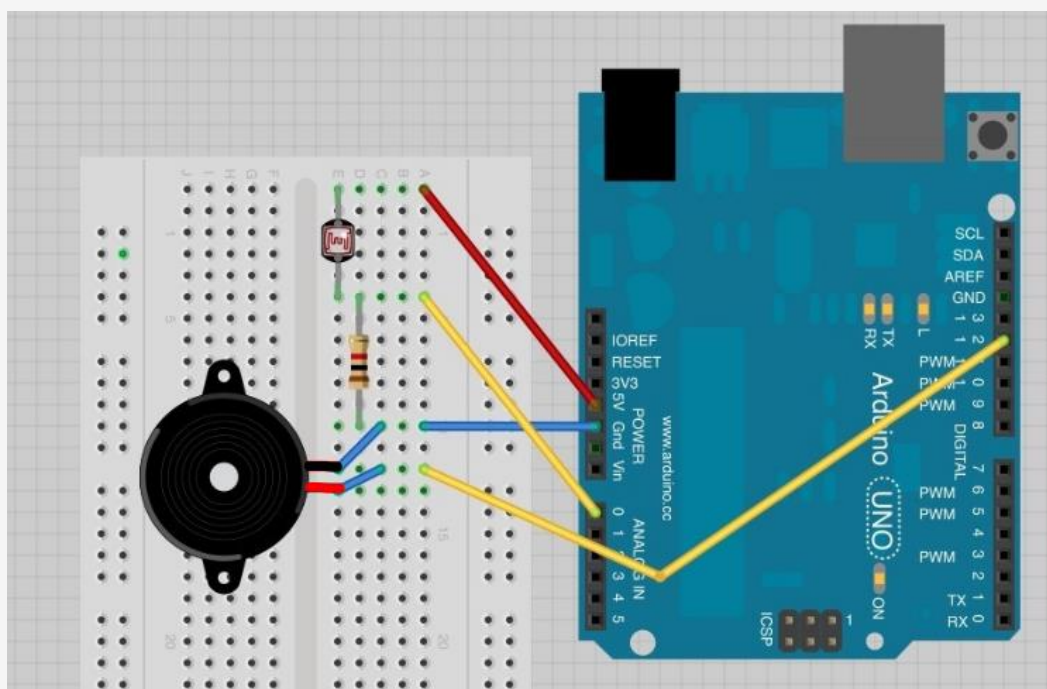
Teremin je sintesajzer koji stvara zvuk dok mašete rukama ispred njega. To je u osnovi elektronički glazbeni instrument.

U ovom projektu napraviti ćemo sličan instrument koji će mijenjati visinu tona dok mašete rukom ispred njega.

Trebat će nam piezo zujalica, svjetlosni senzor i otpornik od 1k ohma.

#### Korak 1: Ožičenje

Dovršite ožičenje prema donjem dijagramu



Slika 50 – Povezivanje teremina s Arduinoom

Izvor: <https://learn.adafruit.com/adafruit-arduino-lesson-10-making-sounds/pseudo-theramin>

#### Korak 2: Kôd



CodER projekt je sufinanciran sredstvima programa Europske unije Erasmus+ te će se provoditi od prosinca 2021. do studenog 2023. godine. Ova publikacija i sav njen sadržaj izražava isključivo stajalište njenih autora i Komisija se ne može smatrati odgovornom prilikom uporabe informacija koje se u njoj nalaze.  
(Broj projekta: 2021-1-FR02-KA220-YOU-000028696)



Sufinancira  
Europska unija

```

int speakerPin = 12;

int photocellPin = 0;

void setup()
{
}

void loop()
{
  int reading = analogRead(photocellPin);

  int pitch = 200 + reading / 4;

  tone(speakerPin, pitch);
}

```

Skica je zapravo vrlo jednostavna. Jednostavno uzimamo analognu očitavanje od A0, za mjerenje intenziteta svjetlosti. Ova će vrijednost biti u rasponu od 0 do 700.

Ovoj sirovoj vrijednosti dodajemo 200 kako bismo 200 Hz učinili najnižom frekvencijom i jednostavno dodamo očitavanje podijeljeno s 4 ovoj vrijednosti kako bismo dobili raspon od oko 200 Hz do 370 Hz.

Kako biste proizveli različite efekte, možete pokušati promijeniti vrijednost za koju se dijeli očitavanje svjetlosnog senzora. Na primjer, zamijenite 4 s 2 i pogledajte što će se dogoditi.

### 3.7.3. Roboti osjetljivi na boje

Kao što naziv govori, sortiranje boja sastoji se od sortiranja objekata prema njihovoj boji.

Očito, to se može postići gledanjem na svaki objekt i odlučivanjem na koje ga mjesto želimo postaviti, međutim, kada objekata postane previše, to može biti zamoran i nevjerojatno ponavljajući zadatak. Tada vam mogu vrlo dobro doći automatski strojevi za sortiranje boja.

Ovi strojevi imaju senzore za boje koji mogu osjetiti boju bilo kojeg predmeta. Nakon detekcije boje, motor ili sustav motora hvata predmet i stavlja ga u odgovarajuću posudu. Strojevi za sortiranje boja mogu se koristiti u različitim područjima gdje su važni identifikacija boja, razlikovanje boja i sortiranje boja. Neka od područja primjene uključuju poljoprivrednu





industriju (sortiranje žitarica na temelju boje), prehrambenu industriju, industriju dijamanata i rudarstva, recikliranje itd.

## Industrijski strojevi za sortiranje boja

Pogledajmo neke industrijske strojeve za sortiranje boja.



Slika 51 - Industrijski strojevi za sortiranje boja

Izvor: <http://hugeacademy.com>

Razvrstavači se mogu podijeliti na sortirke u boji s žlijebom i trakaste.

Remenski sortirnici u boji razbijaju manji postotak materijala (važno za matice) i proizvod ostaje relativno statičan tijekom procesa transporta dok se kreće vodoravno na remenu. U tipu žlijeba materijal klizi po žlijebu zbog gravitacije uzrokujući sudar, trenje i veća okomita kretanja, čime se pogoršava omjer slomljenog materijala. Struktura remena čini prijenos glatkim i stabilnim bez odbijanja materijala.

Razvrstavači u boji s žlijebom su češći posebno za hranu jer su cijene niže, kapaciteti veći i proizvodi se lakše vide s obje strane, što je važno kada oljušteno zrno ima ljusku samo s jedne strane. Razvrstavači u obliku žljebova obično su primjenjivi na određene proizvode, jer je žlijeb dizajniran s posebnim kanalima za ovu vrstu materijala na temelju veličina i oblika materijala. Na primjer, žljebovi od 5 mm koriste se za rižu, žitarice i plastične granule. Ravni žljebovi su dobri za plastične pahuljice, kao što su PET ili pahuljice iz boce za mlijeko.

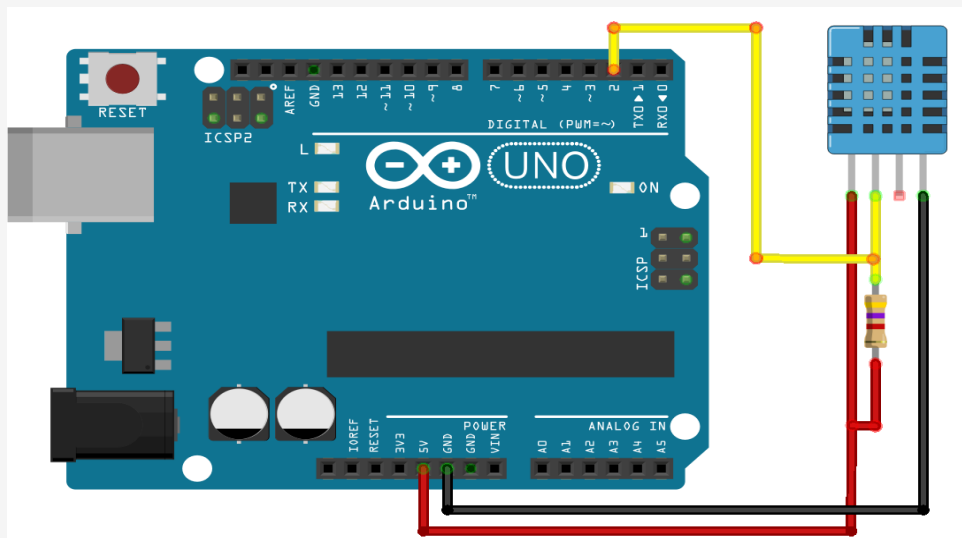
### 3.7.4. Uvod u DHT11 senzor

Postoji senzor za gotovo sve, uključujući i mjerenje temperature!

DHT11 senzor je senzor temperature i vlage, što znači da može mjeriti oba parametra. U ovom projektu učimo kako spojiti DHT11 senzor na Arduino ploču i kako programirati ploču kako bismo vidjeli vrijednosti temperature koje senzor bilježi u stvarnom vremenu.

### Korak 1: Ožičenje

Spojite sve komponente kako je prikazano na donjoj slici:



Slika 52 - DHT11 senzor na Arduino UNO

Izvor: <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>

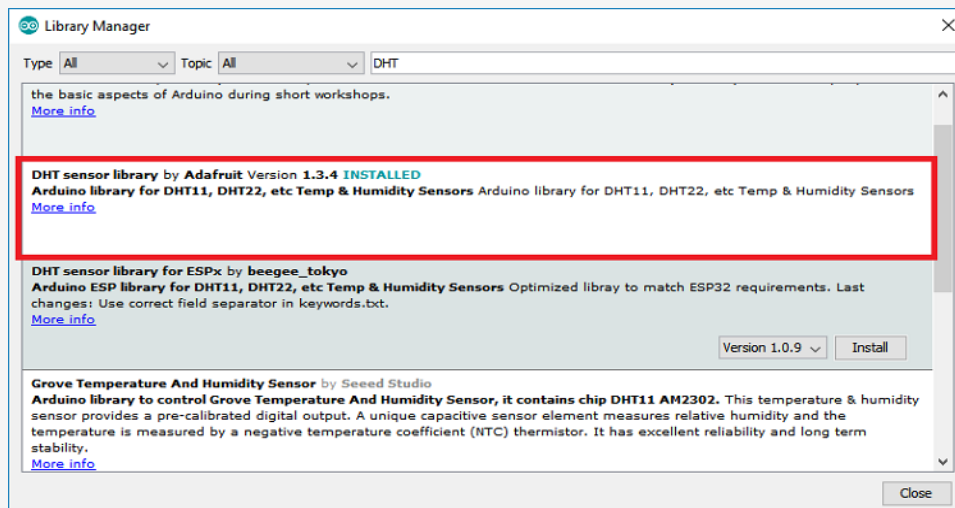
Moguće je zanemariti otpornik na dijagramu, a to je otpornik od 4,7 k ohm.

### Korak 2: Instaliranje knjižnica

Za čitanje s DHT senzora, koristit ćemo DHT biblioteku iz Adafruita. Za korištenje ove knjižnice također morate instalirati Adafruit Unified Sensor knjižnicu. Slijedite sljedeće korake za instaliranje knjižnice.

Otvorite svoj Arduino IDE i idite na Sketch > Include Library > Manage Libraries. Upravitelj knjižnice trebao bi se otvoriti.

Potražite "DHT" u okviru za pretraživanje i instalirajte DHT knjižnicu iz Adafruita.



Slika 53 – Instaliranje DHT knjižnice iz Adafruit

Izvor: <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>

### Korak 3: Kôd

```
#include "DHT.h"

#define DHTPIN 2 // na koji smo pin spojeni

#define DHTTYPE DHT11 // DHT 11 // Inicijaliziraj DHT senzor za normalan 16mhz Arduino

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");

  dht.begin();
}

void loop() {
  // Pričekajte nekoliko sekundi između mjerenja
  delay(2000);
```

```
// Očitavanje temperature ili vlažnosti traje oko 250 milisekundi!
// Očitavanja senzora također mogu biti 'stara' do 2 sekunde (to je vrlo spor senzor)

// Očitaj temperaturu kao Celzijuse
float t = dht.readTemperature();

Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C ");
}
```

### 3.7.5. Izradite pametni ventilator za hlađenje

U ovom projektu spojite svoje znanje o senzorima i aktuatorima kako biste stvorili korisnu i jedinstvenu napravu: inteligentni ventilator za hlađenje. Koristite istosmjerni motor koji pokreće štit motora (to smo istražili u prethodnom poglavlju) uz DHT11 senzor za snimanje temperature vaše sobe.

Konačni rezultat je ventilator za hlađenje koji radi kada se postigne određeni temperaturni prag, naravno da ćete moći programirati sustav kako želite i sami odrediti prag temperature.

#### Korak 1: 3D ispis ventilatora za hlađenje

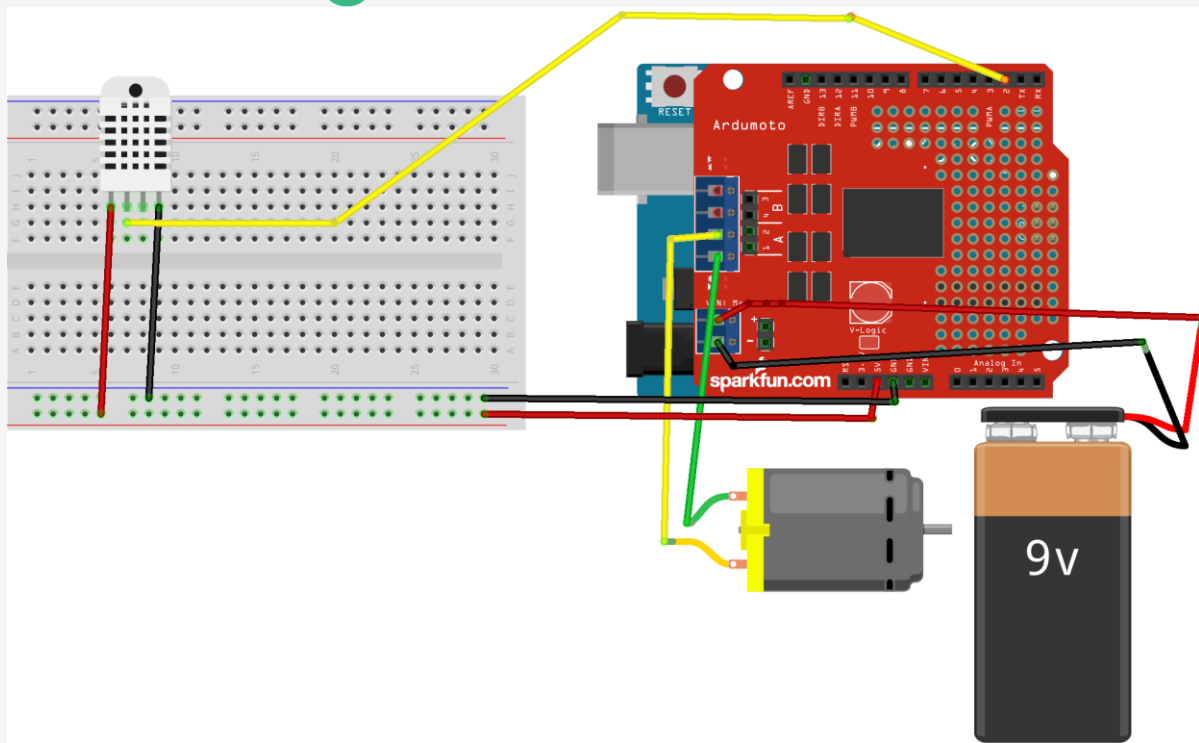
Pronašli smo [ovaj](#) dizajn na thingiverse. To je mini ventilator za hlađenje koji koristi DC motor. Također je moguće ispisati kućište baterije za smještaj 9v baterije 3D metodom. Međutim, to neće biti potrebno za naše potrebe jer ćemo koristiti 9v bateriju za napajanje štita motora.

Probajte isprintati ventilator i podršku. Ako nemate pristup 3D pisaču, naravno, moguće je izraditi ova dva predmeta od kartona ili šperploče ili bilo kojeg drugog materijala koji je dovoljno konzistentan.

#### Korak 2: Spajanje svega zajedno

Spojite sve komponente prema donjoj slici:





Slika 54 – Ožičenje pametnog ventilatora za hlađenje

Izvor: [Digijeunes](#)

### Korak 3: Kôd

Ovim programom nalažemo Arduino ploči da pokrene DC motor ako je temperatura koju bilježi DHT11 senzor jednaka ili veća od 24°. U suprotnom, DC motor se neće okretati.

```
#include "DHT.h"
```

```
#include "AFMotor.h"
```

```
#define DHTPIN 2 // na koji smo pin spojeni
```

```
AF_DCMotor motor1(1); // Na motornom štitu definiram motor pričvršćen na M1
```

```
#define DHTTYPE DHT11 // DHT 11
```

```
// Inicijaliziraj DHT senzor za normalan 16mhz Arduino
```

```
DHT dht(DHTPIN, DHTTYPE);
```

```
void setup() {  
  Serial.begin(9600);  
  
  motor1.setSpeed(100); // Definiramo brzinu kojom će se motor okretati  
  
  dht.begin();  
}  
  
void loop() {  
  // Pričekajte nekoliko sekundi između mjerenja.  
  delay(2000);  
  
  // Očitavanje temperature ili vlažnosti traje oko 250 milisekundi!  
  // Očitavanja senzora također mogu biti 'stara' do 2 sekunde (to je vrlo spor senzor)  
  
  // Očitaj temperaturu kao Celzijus  
  float t = dht.readTemperature();  
  
  Serial.print("Temperature: ");  
  Serial.print(t);  
  Serial.print(" *C ");  
  
  if (t >= 24)  
  {  
    motor1.run(BACKWARD);  
  }  
  else  
  {  
    motor1.run(RELEASE);  
  }  
}
```



```
}
}
```

## Reference

- Autodesk, Inc. (2020). *What You'll Learn*. <https://www.instructables.com/Tools-andMaterials-for-Arduino/>
- Boxall, J. (2013). *Arduino workshop: A Hands-On introduction with 65 projects*. No Starch Press.
- Circuit Basics (n.d.). *Introduction to Microcontrollers*. <https://www.circuitbasics.com/introduction-to-microcontrollers/>
- Green Steam Incubator. (2019). Module on Microcontrollers: 30 hours lessons. <https://steam-incubator.org/wp-content/uploads/2021/11/IO3.2-GSI-Module-on-Microcontrollers.pdf>
- Makerspaces.com (2022). *Arduino For Beginners*. <https://www.makerspaces.com/arduino-uno-tutorial-beginners/>
- Techatronic (2022). *Types of Arduino Boards*. <https://techatronic.com/types-of-arduino-boards-arduino-uno-mega-mini-specification/>
- Learn Adafruit (2022). *Adafruit motor shield*. <https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>
- Sparkfun (2022). *Servos*. <https://www.sparkfun.com/servos>
- Learning about electronics (2022). *555 timer pinout*. <http://www.learningaboutelectronics.com/Articles/555-timer-pinout.php>
- Girls in STEM (2022). EU funded project. <https://girlsinstem.eu/>
- Arduino Get Started (2022). *Arduino light sensor*. <https://arduinogetstarted.com/tutorials/arduino-light-sensor>
- Adafruit (2012). *Pseudo theremin*. <https://learn.adafruit.com/adafruit-arduino-lesson-10-making-sounds/pseudo-theramin>
- Huge academy (2022). <http://hugeacademy.com/>
- Random nerd tutorials (2022). *Complete guide for DHT11 humidity and temperature sensor with Arduino*. <https://randomnerdtutorials.com/complete-guide-for-dht11dht22-humidity-and-temperature-sensor-with-arduino/>
- DIY robotics (2020). *Articulated robots*. <https://diy-robotics.com/article/articulated-robots/>
- DIY robotics (2020). *What you should know about cartesian robot*. <https://diy-robotics.com/article/what-you-should-know-about-cartesian-robot/>
- DIY robotics (2020). *Scara robots*. <https://diy-robotics.com/article/scara-robots/>
- DIY robotics (2020). *Articulated robots*. <https://diy-robotics.com/article/top-six-types-industrial-robots-2020/>
- Learn mech (2022). *Cylindrical robot diagram construction applications*. <https://learnmech.com/cylindrical-robot-diagram-construction-applications/>



How to robot (2022). *Industrial robot types and their different uses*. <https://www.howtorobot.com/expert-insight/industrial-robot-types-and-their-different-uses>

Robot Worx (2022). *What are the main types of robots*. <https://www.robots.com/faq/what-are-the-main-types-of-robots>

Built In (2022). *Robotics*. <https://builtin.com/robotics>

Analytics Insight (2021). *Common types of robots*. <https://www.analyticsinsight.net/common-types-of-robots-are-there-any-new-ones-you-havent-heard-yet/>

Stanford Edu (2022). *Army robots* <https://cs.stanford.edu/people/eroberts/cs201/projects/2010-11/ComputersMakingDecisions/army-robots/index.html>

NCBI (2019). *Articles*. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6625162/>

Guarforce (2022). <https://www.guardforce.com.hk/>

Iberdrola (2022). *Educational robots*. <https://www.iberdrola.com/innovation/educational-robots>







**Sufinancira  
Europska unija**

CodER projekt je sufinanciran sredstvima programa Europske unije Erasmus+ te će se provoditi od prosinca 2021. do studenog 2023. godine. Ova publikacija i sav njen sadržaj izražava isključivo stajalište njenih autora i Komisija se ne može smatrati odgovornom prilikom uporabe informacija koje se u njoj nalaze.

Broj projekta: 2021-1-FR02-KA220-YOU-000028696

