

Te (in večino prihodnjih) zapiskov lahko pregledujete ter začasno spreminjate in poganjate programe v njih tudi tako, da (ne da bi morali kaj namestiti) uporabite Binder.

## Zaganjanje Notebooka

Za začetek bomo Python zagnali in uporabljali malo drugače kot kasneje. Odprli bomo Jupyter notebook, ki ste si ga (upam) namestili.

Odpremo ... bogve, kako je to po slovensko ... ukazna lupina? Pač Cmd, Shell, Konsole, Terminal, ITerm, bash, zsh ali kakorkoli se že to imenuje na vašem operacijskem sistemu in vašem računalniku). Pokazal se bo "ukazni pozivnik" (uh: *prompt*), kjer bo izpisano ime nekega direktorija (po novem moderno imenovanega "mapa") na vašem disku. Če ste na Windowsih in ta direktorij ni vaš "domači direktorij", se pravi, ne vsebuje vašega imena, vpišite

```
cd \Users\benjamin
```

vendar samo, če vam je (na tem računalniku) ime Benjamin. Če vam je kako drugače, vpišite kaj drugega. Če ne uporabljate Windowsov, pa je najbrž itak vse v redu.

```
jupyter notebook
```

izpisala se bo ta in ona nepomembna reč, potem pa se bo odprl brskalnik z vsebino vašega domačega direktorija.

Okna z ukaznim pozivnikom ne zaprite, saj bi s tem umrla tudi naša beležnica. Če vas moti, pa ga lahko zmanjšate.

Zgoraj desno poiščemo tipko **New**, kliknemo in izberemo **Python 3**. Odpre se prazna beležnica.

## Moj računalnik je lahko tudi kalkulator

Vtipkajmo  $1 + 1$  in pritisnimo Shift in Enter.

```
1 + 1
```

```
2
```

Tole bi znal izračunati vsak z minimalnim znanjem matematike. Poskusimo lahko tudi kaj bolj zapletenega.

```
8 * 7
```

```
56
```

```
1+2 * 3+1
```

```
8
```

Nič posebnega. Temu, kar smo vpisovali, pravimo *izraz*. Pri tem predmetu se sicer ne bomo ukvarjali s pravorečjem; teoretiki nas učijo o razliki med *izrazom*, *stavkom* in bogvečem še, sam pa se tule enkrat za vselej opravičujem, ker bom dal prednost praksi, ne terminologiji. Še definicijo pojma *izraz* bom prepustil kolegu Slivniku, ki bo nekatere od vas v tretjem letniku učil zanimivo snov o jezikih, prevajalnikih in sploh vsem).

Izraz bo za nas pač *nekaj, kar se da izračunati* (kar je ravnokar zaropotalo, je neki Alan Turing, ki se je z jabolkom v ustih obrnil v grobu). In gornje stvari se očitno dajo izračunati. In niti niso preveč zanimive.

Razen, morda, zadnjega izraza. V izrazih lahko uporabljamo presledke in to moramo - kot tudi sicer - početi po občutku. V zadnjem izrazu tega nismo počeli, zato je zapis zavaajajoč. Python je dovolj pameten, da ve, da ima *operator* (še ena beseda, ki si jo zapomnimo!) množenja prednost pred operatorjem seštevanja. Kar smo napisali, se računa enako, kot če bi rekli

```
1+2*3+1
```

8

ali

```
1 + 2*3 + 1
```

8

ali

```
1 + 2 * 3 + 1
```

8

Nekoč se bomo dogovorili, da bomo uporabljali zadnjo obliko, dotlej pa postavljajte presledke, kamor vam veva občutek, le nekaj se zmenimo: nikoli jih ne dajajte na začetek. Dokler vam naslednji teden ne bom rekel, da jih morate. Boste videli, zakaj.

Kakšni operatorji še obstajajo? Očitna sta še / za deljenje in - za odštevanje. Dvojna zvezdica, \*\*, pomeni potenciranje (mednju ne smemo napisati presledka, \*\* je kot, recimo, ena *beseda*). Operator % izračuna ostanek po deljenju.

```
5 ** 2
```

25

```
3 ** 4
```

81

```
13 % 5
```

3

Potenciranje ima prednost pred vsem, kar poznamo doslej. Kadar je treba, pa lahko uporabimo oklepaje.

```
(4 + 5) * 4
```

36

Če v isto celico zapišemo več računov, Notebook pokaže samo rezultat zadnjega.

```
1 + 3  
5 * 6  
8 - 2
```

6

Čisto druga reč pa je, če v vrstici ostane kak nezaprt oklepaj. V tem primeru Python ve (in pričakuje), da se izraz nadaljuje v naslednji vrstici.

```
(4 + 2 * (3  
+ 8) -  
2)
```

24

Tule ni lepo, a kdaj drugič nam bo prišlo še zelo prav, boste videli.

Pri množenju je nujno uporabiti zvezdico. Se pravi, pisati moramo  $7 * (2 + 3)$  in ne  $7(2 + 3)$ .

Za deljenje imamo poleg operatorja  $/$  tudi  $//$ , ki deli celoštevilsko.

```
7 / 2
```

3.5

```
7 // 2
```

3

7 gre v 2 trikrat, in ena ostane. A celoštevilskega deljenja ostanek ne zanima.

Celoštevilsko lahko delimo tudi necela števila:

```
4.5 / 1.2
```

3.75

```
4.5 // 1.2
```

3.0

Pozornejšemu je padlo v oči še nekaj zanimivega: če delimo 7 z 2 dobimo 3, če 4.5 z 1.2, pa 3.0. Za prvo predavanje bi lahko to tudi preskočili, a se vseeno pomudimo: Python loči med celimi in necelimi števili. Lepše kot pri deljenju to pravzaprav vidimo pri seštevanju.

```
2.3 + 1.7
```

4.0

2 + 2

4

Če seštejemo realni števili 2.3 in 1.7 dobimo realno število 4.0. To število je sicer "celo", vendar je tako zgolj "slučajno". Rezultat seštevanja dveh realnih števil je realno število, in četudi je celo, ga bo Python "interno" shranil kot necelo število in tudi pri izpisu dodal .0. Da se ve. Če seštejemo celi števili 2 in 2, dobimo celo število 4.

Podobno je bilo pri deljenju. Rezultat (celoštevilskega) deljenja necelih števil je necelo število, rezultat celoštevilskega deljenja celih števil pa je celo število.

Kako hitro napredujemo! To, kar smo pravkar spoznali, so "podatkovni tipi". Točneje, spoznali smo dva podatkovna tipa *cela števila* in *števila s plavajočo vejico*. V angleščini se jima reče *integer* in *floating point number* v Pythonu pa `int` in `float`. Odkod ta čudna imena boste nekateri izvedeli pri kakem drugem predmetu.

Vsaka reč v Pythonu je reč nekega tipa, in če je ta reč število, je bodisi tipa `int` bodisi `float`. (So števila lahko še kakega drugega tipa? Lahko, nekateri jeziki imajo celo kupe številskih tipov. Vendar nas za zdaj ne brigajo.)

Neučakanega študenta morda pograbila radovednost. Kateri podatkovni tipi pa še obstajajo - razen številskih? Le malo naj počaka, kmalu bodo na vrsti.

Preden gremo naprej, samo opozorimo, kaj vas čaka v večini drugih jezikov: v skoraj vseh drugih jezikih deljenje celih števil vrača celo število; v Javi, ki se jo boste učili v drugem semestru, bo  $7 / 2$  enako 3. Obratno opozorilo, seveda, velja za tiste, ki že znate programirati in ste morda vajeni drugače: v Pythonu izraz  $7 / 2$  vrne 3.5. Je to pametno? Najbrž je: ko delimo, večinoma hočemo "pravo" deljenje in le redko celoštevilskega. Še več, pri programiranju velikokrat naredimo napako, ko brezskrbno delimo, ne da bi pomislili na to, s kakšnimi števili - celimi ali ne - delamo. Avtorji Pythona so se zato odločili, naj bo deljenje vedno "pravo", kadar hočemo celoštevilsko, pa moramo to posebej povedati tako, da namesto `/` uporabimo `//`.

## Moj računalnik je lahko tudi kalkulator - s spominom

V Notebooku je preprosto uporabiti rezultat zadnjega izračuna.

2 + 3

5

(2 + \_) \* \_

35

Vidimo? Tam kjer napišemo `_`, se "vstavi" tisto, kar se je izpisalo nazadnje. Tako `(2 + _) * _` v bistvu pomeni `(2 + 5) * 5`. To lahko nadaljujemo.

```
_ + 2
```

```
37
```

Zdaj je bil `_` enak 35. In zdaj je 37; to lahko vidimo tudi tako, da napišemo "račun", ki ne vsebuje drugega kot `_`. :)

```
-
```

```
37
```

Kako pa pridemo nazaj, do rezultata izračuna `2 + 3`, nekaj celic višje? En način, ki pa ni najbolj praktičen, je, da uporabimo številko celice. Zgoraj piše `In [19]` in `Out[19]`, torej gre za celico 19. Njen rezultat je v

```
_19
```

```
5
```

Torej lahko pišemo stvari kot

```
_19 + _20
```

```
40
```

Kar počnemo zdajle, je posebnost Notebooka in nima veliko zveze s Pythonom (ali kakim drugim programskim jezikom). Pa še nepraktično je.

Če hočemo shraniti rezultat nekega izračuna za kasnejšo rabo, to storimo tako:

```
x = 2 + 3
```

Temu pravimo *prirediveni stavek*. Rezultat izraza `2 + 3` smo poimenovali `x`.

Tole zdaj je najbolj pomembna stvar v današnjem predavanju. Ne spreglejte je, čeprav je slišati trivialna, da ne bo kdaj kasneje postala kamen spotike (predvsem tistim, ki že znate programirati in si pod prirejanjem predstavljate nekaj drugega, kot v resnici je).

Da hočemo nekaj prirejati, povemo z enačajem (očitno). Na njegovi desni je nek izraz - torej nekaj, kar se da izračunati. Včasih bo to `2 + 3`, včasih bo kaj veliko bolj zapletenega, pogosto pa bo na desni strani samo številka, kot, recimo, v prirejanju `x = 42`. Python bo, tako kot v našem dosedanem igranju z njim, izračunal tisto na desni in dobil 5 ali 42 ali karkoli že.

Na levi strani enačaja je neko ime. Python bo temu imenu (recimo `x`) priredil tisto, kar je izračunal.

OK? Prireditveni stavek priredi imenu na levi strani enačaja rezultat izraza na desni.

Levo in desno od enačaja praviloma pišemo presledke, zaradi preglednosti.

Python tokrat ni izpisal ničesar v odgovor. Rezultat si je le zapomnil, shranil ga je pod imenom `x`. Kaj lahko počnemo s tem `x`? Lahko ga uporabljamo v drugih izrazih.

```
x + 7
```

```
12
```

```
x ** 2
```

```
25
```

```
13 % x
```

```
3
```

```
x
```

```
5
```

Kadar rečem `x`, Python poišče, tisto, kar je priredil `x`-u. Če smo `x`-u priredili 5 in rečemo `x + 7`, je to isto, kot če bi rekli `5 + 7`.

Temu `x` Slovenci ponosno rečemo *spremenljivka*. Angleži temu namreč pravijo *variable*, Madžari pa *változó*. (Predvsem slednje ni posebej pomembno in ne pride v poštev kot izpitno vprašanje, čeprav ne bo nobene škode, če veste. Koristi pa pravzaprav tudi ne. ;))

(Medklic: da, temu bom govoril *spremenljivka*, a bolj prav bi bilo uporabljati besedo *ime*. Tudi Python temu reče *name*. Besedo *spremenljivka* bom uporabljal, ker smo programerji tako navajeni in železno srajco navade je težko sleči. Čez nekaj mesecev pa bomo izvedeli, kakšna je razlika in zakaj se sploh spotikati ob to.)

Spremenljivko lahko seveda uporabljamo tudi za računanje novih spremenljivk.

```
y = x + 2
```

```
y
```

```
7
```

Vidite, kaj smo storili tu? V prvi vrstici smo priredili vrednost `y`-u, v drugi smo to vrednost izpisali.

Spremenljivka pri programiranju (v večini jezikov) ne pomeni istega kot v matematiki. Spremenljivke v matematiki se, roko na srce, pravzaprav ne spreminjajo. V matematiki  $x$  ne more biti v eni vrstici 5, v naslednji pa 8. Pri programiranju pa lahko.

```
x = 5
```

```
x
```

```
5
```

```
x = 8
x
8
```

Še huje. Če matematiki ne bi znali programirati (pa navadno znajo in to dobro), bi jih utegnilo pretresti tole:

```
x = 5
x = x + 2
x
7
```

Napišite `x = x + 2` pred profesorjem matematike, če si upate!

Kako je  $x$  lahko enak  $x + 2$ ? Saj ni. Ta enačaja ne predstavlja *enakosti*, kot v matematiki, temveč *prirejanje*. Ja? V drugi vrstici Python izračuna vrednost izraza `x + 2`, to je, 7, in to priredi imenu, spremenljivki `x`. Izraz `x = x + 2` torej pomeni, preprosto, povečaj `x` za 2.

Tiste, ki že znajo vsaj malo programirati v kakem drugem jeziku, je morda zmotilo, da spremenljivk nismo nikjer *deklarirali*. (Tisti, ki jim beseda *deklarirati* ne pomeni ničesar, naj ostanejo v umestni nevednosti.) V Pythonu tega (skoraj) ne moremo narediti. Spremenljivka (točneje, ime) se pojavi, ko jo uporabimo, in izgine, ko je ne potrebujemo več.

Tem, ki že znajo programirati - posebej, če so uporabljali kak C# ali Javo - povejmo še, da se Pythonove spremenljivke tudi sicer bistveno razlikujejo od spremenljivk, ki so jih vajeni od ondod. Predvsem ni dobro, da so vas najbrž učili, da so "*spremenljivke kot nekašne škatlice*". Če imamo

```
x = 42
y = x
```

imamo v nekaterih jezikih dve škatlici (in v obeh številko 42), v nekaterih jezikih, med katerimi je tudi Python, pa imamo eno samo škatlico z dvema imenoma, `x` in `y`. Več o tem bomo izvedeli kasneje, ko bomo to zmožnejši prebavljati in bomo videli tudi konkretno zmotnost ideje, da je "*spremenljivka kot nekašna škatlica*".

Ostali, tisti, ki niso še nikoli programirali, pa se sprašujejo nekaj drugega. Kakšna so lahko imena spremenljivk? Vedno le ena črka? Angleške abecede?

Takole: imena (angleško govoreči jim pravijo *identifier*) so lahko poljubno dolga. Vsebujejo lahko črke angleške abecede, številke in podčrtaj, `_`, vendar se morajo začeti s črko ali podčrtajem, s številko pa ne. Python, kot skoraj vsi drugi jeziki, razlikuje med malimi in velikimi črkami: `x` in isto kot `X`. (V resnici smemo uporabljati tudi šumnike in cirilico in kitajske pismenke, vendar se tega ne navadite, ker je nezaželeno in ker tega ne boste mogli početi v skoraj nobenem drugem jeziku kot v Pythonu, zato naj vam ne pride v kri.)

Poleg tega obstaja še par dogovorov. Vsak jezik ima namreč poleg obveznih pravil tudi želeni slog. (Razen nekaterih, ki jih imajo več; najbolj notorično brezvladje je najbrž v C in C++.) Predvsem pa - in to vas bo kot računalnikarje prej ko slej zadelo - vsako resno podjetje določi pravila oblikovanja kode. Google, recimo, jih je predpisal za vse jezike, ki jih pogosteje uporablja (Google Style Guides). Vsi, ki programirajo v tem podjetju, se morajo teh pravil držati.

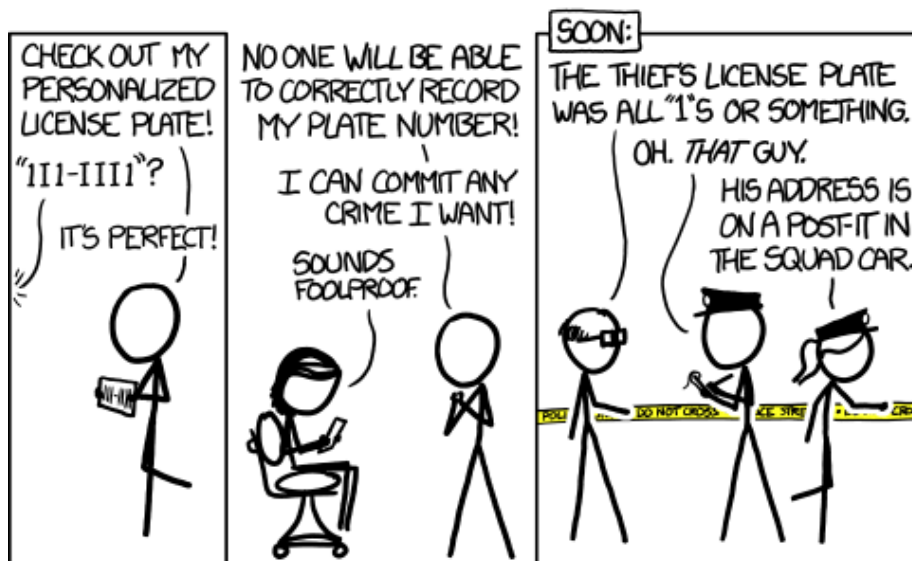
1. **Imena naj se vedno začnejo z malo črko.** Nekateri ste se učili drugih jezikov, kjer so navade drugačne. Ali pa so vas učili Python učitelji, ki nikoli niso brali pravil za Python. Kasneje bomo spoznali določene stvari, ki jih poimenujemo z veliko začetnico; dotlej pa - mala.
2. **Če je ime sestavljeno iz več besed, jih ločimo s podčrtajem,** , na primer `velikost_cevlja`. Obliko `velikostCevlja` boste videli le v starejših programih v Pythonu.
3. **Ime naj pove vsebino, ne vrste spremenljivke,** npr. `pospesek = 9.8`. Spremenljivke ne poimenujte `stevilka`, ceprav morda res vsebuje številko. Ko bomo začeli uporabljati sezname, vas bo veliko uporabljalo ime `seznam`, in če bodo v programu trije sezname, jih obsto poimenovali `seznam1`, `seznam2`, `seznam3`. Da, to vam bo zelo pomagalo razumeti, kaj ste shranili kam. To je približno tako, kot če bi na knjigah pisalo "Knjiga" namesto, kaj je v njej. (Ni pa zelo drugače od steklenic v Mercatorju, na katerih preprosto piše "Vino".)
4. **Kratka imena.** Izogibajte se imen, kot so `a`, `aa`, `asd...` ki ne povedo nič. Sicer me boste videli, da bom uporabljal kratka imena na predavanjih, vendar le zato, ker bom tipkal v živo in nimate časa gledati, kako tipkam `imena_nekih_studentov`. Program, v katerem je deset spremenljivk z imeni `a`, `b`, `aa`, `a2`, `a3`, `a4` je nemogoče brati.
5. **Dogovorjena imena spremenljivk.** Po drugi strani pogosto uporabljamo kratka imena za "nepomembne" spremenljivke, ki "živijo" le nekaj vrstic. Kot v matematiki bodo tudi tu spremenljivke z imeni `i`, `j`, `k` pogosto števci, se pravi, nekaj, kar bo teklo od, recimo, 1 do 10. Po eni strani neinformativno, po drugi pa vedno, ko vidimo `i`, `j`, `k` vemo, da gre za števec. `e` bo element kakega zaporedja. `x` in `y` bosta argumenta (tako kot matematiki vedno rečejo `sin(x)`). Imena `s`, `t`, `u` bodo pogosto sezname. Nekateri imajo za seznam raje `xs` (kot angleška množina imena `x`), saj je `xss` potem seznam, ki vsebuje sezname. To boste videli predvsem v nalogah na vajah. Ime `s` bo sicer pogosto predstavljalo tudi besedilo.

Tega se vam niti slučajno ni potrebno zapomniti. Glejte, kako bom pisal na predavanjih in v zapiskih, pa se boste navadili.

Na to temo bi se dalo povedati še veliko in morda nekoč bomo. Za zdaj pa vam resno polagam na srce, kar je napisano tu. Predvsem tistim, ki že znate programirati in se vam zdi najlepše tako, kot delate zdaj. Meni se zdijo v Pythonu lepa Pythonova pravila, v JavaScriptu pa pravila JavaScripta.

Kdor razmišlja drugače, je kot Francoz, ki iz patriotizma namerno govori angleško s francoskim naglasom. Ali Štajerc, ki jo zavija po štajersko.

Vsi naštetih dogovori (in še veliko podobnih bomo spoznali sproti) so samo dogovori. Če se jih ne držimo, bodo programi še vedno delovali. Pythonu je vseeno. Ni pa vseeno nam: če se držimo takšnih dogovorov, bodo naši programi preglednejši tako za nas, kot za druge, ki upoštevajo enaka pravila pisanja in bodo morali brati naše programe za nami.



## Poglavje polno napak

"Vsak, kdor dela, dela tudi napake," se je jež poklapano opravičil krtači. Nobenega omemba vrednega kosa programa, daljšega od, recimo, 20 vrstic, ne napišemo brez vsaj ene resne napake. Pravzaprav tudi jaz, ki sem kar usposobljen programer, večine časa ne preživim ob programiranju, temveč ob iskanju napak v tem, kar sem pravkar sprogramiral. Začetni tečajji programiranja pa navadno ne posvečajo napakam nobene resne pozornosti. Tu te napake ne bomo ponovili, zato kar takoj naredimo nekaj napak.

Postavimo, najprej,  $a$  na 7 in izračunajmo  $a + b$ .

```
a = 7
a + b
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[35], line 2
      1 a = 7
----> 2 a + b
```

```
NameError: name 'b' is not defined
```

Ker je iskanje napak glavno opravilo programerja, se programski jeziki (točneje: programi, ki izvajajo programe, napisane v programskih jeziki :) trudijo biti karseda prijazni in poskušajo programerju čimbolj jasno povedati, kaj je zamočil.

Ko bomo programirali zares, bomo videli tudi daljša, ki jih bomo težje razumeli, tole pa je preprosto: `name 'b' is not defined`. Pozabili smo *definirati* `b`, pozabili smo mu dati vrednost.

Kaj pa tole?

```
7 = a
```

```
Cell In[36], line 1
```

```
7 = a
^
```

```
SyntaxError: cannot assign to literal here. Maybe you meant '==' instead of '='?
```

Človek, vaje matematike, bi si mislil, da je `a = 7` in `7 = a` eno in isto. V matematiki da, pri programiranju (v normalnih jezikih) pa ne, saj enačaj pomeni prirejanje; v prvem primeru priredimo `a`-ju `7`, v drugem primeru pa hočemo sedmici prirediti `a`, kar seveda ne gre. To ima natanko toliko smisla, kot če bi napisali `1 = 2`. (Še več, Python nas bo po prstih celo, če bomo napisali `1 = 1`. Ena je ena, to bo ostala in se ne bo spremenila, niti v ena ne.) Sporočila o napaki tokrat ne razumemo povsem, saj ne vemo, kaj je "literal", osnovno sporočilo, "can't assign", pa je jasno.

Pridelajmo še eno napako.

```
True = 12
```

```
Cell In[37], line 1
```

```
True = 12
^
```

```
SyntaxError: cannot assign to True
```

Beseda `True` ima poseben pomen in je ni mogoče uporabljati kot spremenljivko. Takšnim besedam pravimo ključne besede, ali, kot bi jim rekel John Kennedy, če bi bil še živ, *keywords*. Tokrat je bil Python še prijazen, pri večini drugih ključnih besed pa ne bo povedal kaj dosti več kot "nekaj je narobe". Poskusimo z dvema, `if` in `in`:

```
if = 7
```

```
Cell In[38], line 1
```

```
if = 7
^
```

```
SyntaxError: invalid syntax
```

```
in = 7
```

```
Cell In[39], line 1
```

```
    in = 7
    ^
```

```
SyntaxError: invalid syntax
```

Sporočilo "invalid syntax" pomeni, da smo napisali nekaj tako čudnega, da Python ne more uganiti, kaj smo mislili in nam lahko le pokaže tisto mesto, na katerem je dokončno obupal nad nami.

Morda je koga zaskrbelo, da nam bodo takšne, rezervirane besede v stalno napoto. Bi se dalo videti spisek? Ne bo hudega. Python 3.7 jih ima 35 (naraščanje je počasno: v zadnjih desetih letih je dobil le dve novi) in zelo hitro bomo mimogrede spoznali in uporabljali skoraj vse. Že od naslednjih predavanj naprej vam ne bo prišlo na misel, da bi uporabili `if` kot ime spremenljivke in še kak teden kasneje vam bo stavek `if = 1` videti grotesken. (Vsaj mene zabolijo oči, ko ga pogledam in prsti se upirajo temu, da bi sploh natipkali kaj takšnega.) Spisek? Ne bom vas strašil z njim, lahko pa si pomagata z Googleom, če hočete. A ni potrebe.

Še dve sintaktični:

```
(2 + 3 * 5))
```

```
Cell In[40], line 1
```

```
    (2 + 3 * 5))
    ^
```

```
SyntaxError: unmatched ')'
```

```
5 +
```

```
Cell In[41], line 1
```

```
    5 +
    ^
```

```
SyntaxError: invalid syntax
```

Pa še ena preprosta:

```
a = 5
```

```
7 / (a - 5)
```

```
ZeroDivisionError
```

```
Traceback (most recent call last)
```

```
Cell In[42], line 2
```

```
    1 a = 5
```

```
----> 2 7 / (a - 5)
```

`ZeroDivisionError: division by zero`

Ko program izpiše napako jo moramo razumeti in *potem* popraviti program. Tule, ko pišemo le čisto kratke izraze, se o tem ni kaj dosti pogovarjati. Ko gre za daljše programe, pa programa, ki ne deluje, temveč javlja kako napako, študenti pogosto ne poskušajo popraviti, temveč ga le spreminjajo, dokler ne začne delovati. To je slabo iz dveh razlogov. Prvi je, da je to neučinkovito. Python se res trudi, po najboljših močeh, da bi povedal, kaj je narobe. Veliko učinkoviteje je razumeti, kaj nam hoče povedati in ukrepati, kot pa brskati kot slepa kura. Sploh zato, ker so programi pri tem predmetu kratki in se jih *mogoče* da popraviti na slepo srečo. V resničnem življenju pa so daljši in je, prosim, verjemite mi, res potrebno razumeti napake.

Drugi razlog, da slepo popravljanje napak ni dobra ideja, je, da utegnemo na ta način le obiti simptome, neposreden vzrok za napako. Prava napaka pa lahko ostaja nekje drugje, skrita, nepopravljena.

Znanje branja napak je osnovna veščina programerja. Pomembno je, da jo obvladate.

## Moj kalkulator ima tudi funkcije

Tako kot spremenljivke, ki v programiranju ne pomenijo čisto istega kot v matematiki in v programiranju ne pomenijo čisto istega, tudi beseda *funkcija* (Rihanna bi rekla *function*, če bi znala programirati, v kar pa dvomim) ne pomeni povsem istega. Videti pa je zelo podobno. Imamo, recimo, funkcijo `abs`, ki izračuna absolutno vrednost števila.

```
abs(-2.8)
```

2.8

Ali pa `pow`, ki naredi isto kot operator `**`.

```
pow(2, 3)
```

8

Ob `pow` omenimo - predvsem za tiste, ki že znajo programirati v kakem drugem jeziku -, da je bolj običajno pišemo `2 ** 3` in ne `pow(2, 3)`. Funkcija `pow` (okrajšava za angleški *power*) ima svoje super powers in uporabljamo jo le takrat, ko jih potrebujemo.

Za razliko od matematikov, ki na funkcijo gledajo, kot da *ima določeno vrednost pri določenih parametrih*, računalnik *izračuna vrednost funkcije*, za kar moramo *poklicati funkcijo*. Se pravi, v zadnji vrstici smo *poklicali* funkcijo `pow` in ji *podali* dva *argumenta*, 2 in 3. Funkcija je izračunala vrednost in jo *vrnila*.

Tudi klic funkcije, `pow(2, 3)`, je izraz. Kot katerikoli drugi izraz lahko tudi `pow` in `abs` nastopata kot del izraza.

```
(pow(2, 3) + 2) / 5
```

```
2.0
```

```
pow(2, 3) + abs(-2)
```

```
10
```

In argumenti funkcij so lahko prav tako izrazi.

```
x = 1
```

```
yy = pow(abs(-2), x * 3)
```

```
yy
```

```
8
```

Funkcije so v resnici zelo pomembna reč. Python ima milijone in milijone funkcij (ne pretiravam, samo en detajl sem zamolčal). Za vsako reč, ki si jo zamislite, obstaja funkcija. Obstaja funkcija, ki bo, če jo pokličete, poslala mail osebi, katere naslov podate kot argument in z vsebino, ki jo podate kot argument. Obstaja funkcija, ki izriše ali pokaže sliko. Funkcija, ki odpre Excel in funkcija, ki piše po njem. S Pythonom lahko naredimo karkoli, le ime funkcije moramo poznati. (Kot rečeno, izpuščam detajle.)

## Nizi

Nas še vedno muči radovednost o tem, kakšne podatkovne tipe, poleg številskih, še imamo? Ker radovednost ni lepa čednost, jo bomo najlažje odpravili tako, da jo potešimo. Vsaj malo. ("*Skušnjavo najlažje premagaš tako, da ji podležeš*," je modroval Oscar Wilde. Res pa je, da so ga kasneje aretirali in zaprli.) Spoznajmo vsaj še en bolj zapleten podatkovni tip: niz.

Niz ali, kot mu pravijo Avstralci, *string* (oprostite mi boste morali, da bom pogosto uporabljal angleške izraze; pa mi pokažite, lepo prosim, zidarja, ki ne uporablja vaservage in plajbe temveč vodno tehtnico in svinčnico, pa se bom tudi jaz discipliniral), je zaporedje znakov. Aha, kaj pa je to znak? Znaki so črke, številke, ločila in take stvari. Nize moramo vedno zapreti v narekovaje, bodisi enojne (') bodisi dvojne ("). Uporabiti smemo take, ki so bolj praktični in tudi Python bo izpisoval tako, kot se mu bo zdelo bolj praktično.

```
'Tole je primer niza.'
```

```
'Tole je primer niza.'
```

```
"Tole je pa še en primer niza."
```

```
'Tole je pa še en primer niza.'
```

Tudi ko smo niz zaprli v dvojne narekovaje, je Python izpisal enojne. V resnici mu je vseeno. (V nekaterih jezikih imajo različni narekovaji povsem ali pa vsaj nekoliko različen pomen. V Pythonu ne, pač pa različne pomene, kot bomo videli, dosežemo tako, da prednje postavimo kak primeren znak.)

Tudi nize lahko priredimo spremenljivkam.

```
napoved = "Jutri bosta matematika pa dež"
napoved
```

```
'Jutri bosta matematika pa dež'
```

Celo seštevamo jih lahko.

```
"Jutri bosta " + "matematika" + " pa " + "dež"
```

```
'Jutri bosta matematika pa dež'
```

Ali pa oboje

```
napoved_zac = "Jutri bosta "
mat = "matematika"
dez = "dež"
napoved_zac + mat + " pa " + dez
'Jutri bosta matematika pa dež'
```

Kako zapleten račun! Predvsem ne spreglejte, da smo dali besedo "pa" pod narekovaje, saj so `napoved_zac`, `mat`, `dez` spremenljivke (ki so v resnici nizi), " pa " pa je niz kar tako. To je nekako tako, kot če bi, ko smo se igrali s številkami, pisali

```
x = 1
y = 3
x + 2 + y
6
```

V zadnji vrstici sta `x` in `y` sta spremenljivki (ki sta v resnici številki), `2` pa je številka kar tako.

Kaj pa, če bi slučajno pozabili narekovaje?

```
napoved_zac + mat + pa + dez
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[54], line 1
----> 1 napoved_zac + mat + pa + dez
```

```
NameError: name 'pa' is not defined
```

Jasno? Brez narekovajev je `pa` ime spremenljivke - in to takšne, ki še ni definirana. To je tako, kot če bi namesto

```
ime = "Benjamin"
```

kar je pravilno, rekli

```
ime = Benjamin
```

```
-----
NameError                                Traceback (most recent call last)
Cell In[56], line 1
----> 1 ime = Benjamin
```

```
NameError: name 'Benjamin' is not defined
```

Ali pa:

```
napoved = Jutri bosta matametika pa dež
```

```
Cell In[57], line 1
    napoved = Jutri bosta matametika pa dež
                ^
```

```
SyntaxError: invalid syntax
```

Tule računalnik trpi še bolj. Ne le, da so *Jutri*, *bosta*, *matematika*, *pa* in *dež* nedefinirane spremenljivke (ker so pač brez narekovajev), računalnik tudi nima pojma, kaj hočemo pravzaprav početi z njimi, čemu mu naštevamo imena nekih spremenljivk. Jih hočemo sešteti ali zmnožiti ali kaj? Kot vedno, kadar računalniku napišemo kaj zares zmedenega, nam zastoka le "syntax error" in pokaže mesto, kjer se je dokončno izgubil.

Gornje je tako, kot da bi rekli

```
x = 1
y = 2
z = 3
x y z
```

```
Cell In[58], line 4
    x y z
        ^
```

```
SyntaxError: invalid syntax
```

Ubogemu računalniku pač ni jasno, kaj bi z *x*, *y* in *z* ter zakaj mu jih naštevamo.

Tiste, ki so ob vpisu na fakulteto nihali med računalništvom in slavistiko, je nemara zbadlo, da sem pisal "Jutri bo matematika pa dež". Mar ne bi bilo lepše (in bolj prav) "matematika in dež". Ponovite vse, kar smo napisali zgoraj, z *in*, pa boste videli, čemu. Za jezikovno okornost se seveda opravičujem, ampak z *in* tale primer ne bi deloval.

Zakaj pa smo prejle rekli, da uporabimo tiste narekovaje, ki so bolj *praktični*? Čemu bi bili kakšni narekovaji bolj praktični od drugih?

```
"Cesar vpraša nekoliko nevoljen: "Kaj neki?""
```

```
Cell In[59], line 1
    "Cesar vpraša nekoliko nevoljen: "Kaj neki?""
```

`SyntaxError: invalid syntax`

Ni potrebno biti posebno pameten, da vidimo, kaj ga je (namreč Pythona, ne cesarja) onesrečilo tokrat. Ko vidi prvi narekovaj, ve, da gre za niz. Ko pride do naslednjega narekovaja, se niz, tako méni, niz konča. In potem se seveda zmede, ker nizu sledi nekaj, kar ni podobno ničemur. Zdaj pa poskusimo z enojnimi narekovaji.

```
'Cesar vpraša nekoliko nevoljen: "Kaj neki?"'
```

```
'Cesar vpraša nekoliko nevoljen: "Kaj neki?'''
```

Ker se niz začne z enojnim narekovajem, se bo s takim tudi končal in vsi dvojni narekovaji znotraj niza so samo znaki kot katerikoli drugi - tako kot recimo dvopičje in vprašaj. O tej temi bi lahko napisali še marsikaj, vendar se za zdaj ustavimo.

## Iz nizov v števila

Da se reč usede, meditirajmo ob naslednjih vrsticah:

```
a = 1 + 1
b = "1 + 1"
c = "1" + "1"
```

Kakšne so po tem vrednosti spremenljivk `a`, `b` in `c`? Sploh pa, je vse troje legalno ali pa bo Python spet kaj stokal?

V prvo nimamo dvomov, vrednost `a` mora biti enaka 2 (in tudi je).

```
a
2
```

Drugo? `"1 + 1"` je niz; spremenljivki `b` smo priredili niz `"1 +1"`, torej vsebuje ta niz. In ne niza 2? Ne, nihče mu ni naročil, naj poskuša izračunati, koliko je `1 + 1`, tako kot pravzprav tudi v `ime = "Benjamin"` ne poskuša izračunati, "koliko" je Benjamin. `"1 + 1"` je niz, kot vsak drugi, čeprav je slučajno podoben računu.

```
b
'1 + 1'
```

Najbolj zanimivo je tretje. Preden razrešimo vprašanje, se vprašajmo nekaj drugega. Recimo

```
ana = "Ana"
benjamin = "Benjamin"
r = ana + benjamin
```

Kaj dobimo, če seštejemo Ano in Benjamin. (Tončka? Brez duhovičenja, to so resne reči.) Spremenljivka `r` bo imela vrednost `"AnaBenjamin"`. Glede tega smo si menda enotni, ne? (Ako kdo misli, da bomo dobili `"Benjamin Ana"`, saj smo tudi poprej imeli posledke ob oni študijsko-vremenski napovedi, naj pozorno pregleda, kaj smo pisali ondi: vse posledke smo napisali sami.)

No, potem pa vemo: ko seštejemo *niza* `"1"` in `"1"` niz `"11"`. `"1"` in `"1"` torej ni `"2"`, temveč `"11"`.

```
c
```

```
'11'
```

Nikar ne zamudimo priložnosti za še eno napako!

```
1 + "1"
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[66], line 1
----> 1 1 + "1"
```

`TypeError: unsupported operand type(s) for +: 'int' and 'str'`

Seštevanje je *operacija*, zato tistemu, kar je levo in desno od nje pravimo *operanda*. Sporočilo pravi, da operator `+` ne podpira operandov tipov `int` in `str` (`str` je podatkovni tip, ki predstavlja nize). Dve števili ali dva niza bi znal sešteti, te kombinacije pa ne. Mimogrede, obratni vrstni red da nekoliko drugačno sporočilo:

```
"1" + 1
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[67], line 1
----> 1 "1" + 1
```

`TypeError: can only concatenate str (not "int") to str`

Stikanje (*concatenation*) je le drugo ime za seštevanje nizov; sporočilo pravi, da ne moremo stakniti niza in števila. Na to napako boste v prvih dveh tednih vaj pogosto naleteli, tako da bodite pripravljeni. Kasneje bomo delali drugačne stvari in bo šlo mimo.

Ker nam bo prišlo vsak čas prav, povejmo, kako iz niza dobimo število. Recimo, torej, da imamo `a = "1"` in `b = "2"`. Radi bi ju sešteli - vendar zares, tako da bomo dobili 3, ne `"12"`. Za to ju moramo najprej (ali pa sproti) spremeniti v števili. Iz niza dobimo število tako, da pokličemo "funkcijo" `int` ali `float`; obe funkciji pričakujeta kot argument niz, ki vsebuje neko število in kot rezultat vrneto celo (`int`) ali realno (`float`) število. (Tole bi se spodobilo in bilo pravično povedati: `int` in `float` v resnici nista funkciji, temveč nekaj drugega, a za

potrebe prvih toliko in toliko predavanj, se bomo držali Pythonovega načela "Če hodi kot raca in gaga kot raca, potem je raca", ki bo postalo pomembno proti koncu semestra; če se obnašata kot funkciji, ju bomo brez slabe vesti oklicali za funkciji.)

```
int("42")
```

```
42
```

```
float("42")
```

```
42.0
```

Kar želimo, storimo na tri načine, vsak bo poučen po svoje. Prvi:

```
a = "1"
```

```
b = "2"
```

```
aa = int(a)
```

```
bb = int(b)
```

```
aa + bb
```

```
3
```

Naredili smo dve novi spremenljivki, `aa` in `bb`, ki vsebujeta vrednosti `a` in `b`, pretvorjeni v števila. Nato ju seštejemo.

Drugi:

```
a = "1"
```

```
b = "2"
```

```
a = int(a)
```

```
b = int(b)
```

```
a + b
```

```
3
```

Tole je podobno kot prej, le da smo povozili stare vrednosti `a` in `b` z novimi, številskimi, namesto da bi števila zapisovali v druge spremenljivke.

Samo za tiste, ki že znate programirati, a ne v Pythonu: po izkušnjah iz preteklih let, se tule začne polovica predavalnice praskati po glavi, pogumnejši pa se začnejo oglašati, da je Python čuden jezik oziroma, z generaciji lastnejšimi besedami, *kr neki*. To, da tipov spremenljivk ni potrebno deklarirati, so še nekako požrli. Ampak tole tule je pa višek, pravijo: je `a` tipa niz ali število? No, na srečo se lahko skličem na to, kar sem polagal v srce ob prireditvenem stavku: prirejanje priredi imenu neko vrednost. Predvsem pa Python nima spremenljivk v takem pomenu besede, kot ste jih vajeni iz Java, C# ali C++, temveč ima imena za objekte. Prirejanje `a = "1"` priredi imenu `a` niz "1". Prirejanje `a = int(a)` priredi imenu `a` številko 1. Ime `a` se po tem pač ne nanaša več na nek niz temveč

na neko število. Python nima spremenljivk, temveč imena. In imena nimajo tipov.

Tretji:

```
a = "1"
b = "2"
```

```
int(a) + int(b)
```

```
3
```

Ker je `int` funkcija, lahko nastopa v izrazu; potrebe, da bi prepisovali številke v kake nove ali stare spremenljivke, niti ni.

## Vpis in izpis

Doslej smo z računalnikom komunicirali tako, da smo tipkali ukaze v *ukazno vrstico* in če je imela reč kak rezultat (kot ima izraz `1 + 1` rezultat 2), ga je računalnik izpisal. Ukazna vrstica nam v Pythonu pogosto pride prav za kakšna preskušanja, ko programiramo zares, pa komunikacija z uporabnikom poteka drugače. Če hočemo, da računalnik kaj izpiše, mu moramo to posebej reči. Se pravi, ko bomo programirali, ne bomo napisali le `1 + 1` temveč "izpiši, koliko je `1 + 1`".

Rekli smo, da funkcije pri programiranju niso nekaj takšnega kot funkcije v matematiki: funkcije v matematiki imajo določeno vrednost pri določenih argumentih (še huje, matematiki pravijo, da so funkcije pravilo, ki vsakemu elementu kodomene funkcije določi ... uh, pustimo). "Naše" funkcije pa nekaj delajo in včasih vrnejo kakšen rezultat, recimo številko ali niz ali kaj tretjega. Ena od teh funkcij je namenjena izpisovanju: če jo pokličemo, izpiše tisto, kar smo ji dali kot argument. Imenuje se `print`. Za razliko od, recimo, `abs`, ki zahteva en argument, namreč poljubno število, in vrne njegovo absolutno vrednost, ali `pow`, ki hoče natanko dva argumenta, lahko damo `printu` poljubno število argumentov - številke, nize ali še kaj tretjega -, pa jih bo lepo izpisala.

```
print(1 + 1, 27, "benjamin")
```

```
2 27 benjamin
```

```
print(napoved_zac, mat, "pa", dez, "in", 18, "stopinj")
```

```
Jutri bosta matematika pa dez in 18 stopinj
```

Med reči, ki jih izpiše, bo `print`, če ne zahtevamo drugače, postavil presledke.

Druga funkcija, ki nam bo prišla prav, prosi uporabnika, da vpiše kako reč. Kot argument pričakuje niz, vprašanje, ki ga želimo zastaviti uporabniku. Kot rezultat "izračuna" vrne niz, ki ga je vpisal uporabnik.

```
geslo = input("Geslo? ")
```

```
Geslo? qwerty123
```

```
geslo
```

```
'qwerty123'
```

## Prvi skoraj čisto pravi program

Sestavimo tole: računalnik naj uporabnika prosi za temperaturo v Celzijevih stopinjah in računalnik mu bo izpisal, koliko je to v Kelvinih in koliko v Fahrenheitih. Iz Celzijev dobimo Kelvine tako, da jim prištejemo 273.15, Fahreneite pa tako, da jih pomnožimo z 9/5 in prištejemo 32 (kogar zanima še kaj, naj pogleda na Wikipedijo).

```
temp_C = input("Temperatura [C]? ")
temp_K = temp_C + 273.15
```

```
Temperatura [C]? 25
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[78], line 2
      1 temp_C = input("Temperatura [C]? ")
----> 2 temp_K = temp_C + 273.15
```

TypeError: can only concatenate str (not "float") to str

*Eh.* Funkcija `input` vrne *niz*, ki ga je vpisal uporabnik. Četudi utegne ta izgledati kot številka, je še vedno niz in k nizom ni mogoče prištevati števil. Kot smo videli, lahko storimo troje: naredimo novo spremenljivko, na primer, `temp_Cf = float(temp_C)`, povozimo staro s `temp_Cf = float(temp_C)` ali pa pretvorbo opravimo kar sproti, tako da računamo `temp_K = float(temp_C) + 273.15`. Izmed naštetih možnosti se odločimo za četrto in niz pretvorimo, čim ga uporabnik vpiše. Ponovimo torej vso vajo.

```
temp_C = float(input("Temperatura [C]? "))
temp_K = temp_C + 273.15
temp_F = temp_C * 9 / 5 + 32
print(temp_C, "C je enako", temp_K, "K ali", temp_F, "F")
```

```
Temperatura [C]? 25
```

```
25.0 C je enako 298.15 K ali 77.0 F
```

## Beležnica, Python in okolja

V začetku smo rekli, da bomo Python za začetek poganjali drugače kot sicer. Na koncu ga le poženimo tako, kot ga bomo sicer, vmes pa se še kaj naučimo.

## Beležnica

Ko smo vtipkali `jupyter notebook` smo na računalniku zagnali lokalni spletni strežnik. Ker je *lokalni*, je dosegljiv le z našega računalnika (da bi bilo drugače, bi se morali posebej potruditi). Ta spletni strežnik omogoča pisanje besedila (kar, recimo, ravnokar počnem) in izvajanje koščkov kode v Pythonu. Beležnica je uporabna, no, kot beležnica. Vanjo si lahko kaj zapišemo, to shranimo, delimo z drugimi, objavljamo na spletu. Vanjo lahko dodajamo slike, znotraj nje lahko (s Pythonom) rišemo grafe.

Pri tem predmetu nam bo služila za zapiske. Če jih boste naložili na svoj računalnik, jih boste lahko dopolnjevali, se igrali s programi, zapisanimi v njih... Programirali pa bomo pretežno v razvojnih okoljih.

## Python

Tega, kar razlagam tu, pri tem predmetu ne bomo počeli. Koristno pa je povedati, da bolj razumemo, kaj se dogaja in kako stvari delujejo.

Programe pišemo v datoteke. Odprimo primeren urejevalnik: vsem priporočam, da si v take namene namestite Visual Studio Code, sicer pa so z MS Windows pride Notepad, na macOS pa, recimo, TextEdit. Razni Word in Pages za to niso primerni, ker datoteke ne shranijo v pravi obliki.

Odprevši urejevalnik vanj skopiramo gornji program.

Reč shranimo pod imenom, recimo, "temperature.py". (Če to resno kdo dela v Notepadu, naj pri shranjevanju dejansko natipka tudi narekovaje, sicer mu bo Notepad shranil temperature.py.txt.)

Spet odpremo ukazno lupino (Cmd, Terminal ...) in vtipkamo `cd` in ime direktorija, kamor smo shranili temperature.py, potem pa (na Windowsih) `c:\python310\python.exe temperature.py` (ali nekaj takega), drugje pa `python3 temperature.py`.

Z `c:\python310\python.exe` oziroma s `python3` poženemo program Python. Program Python zna brati in izvajati programe v Pythonu. S `temperature.py` mu povemo ime datoteke s programom, ki naj ga izvede. Program Python lepo, vrstico za vrstico bere datoteko in počne, kar piše v njej.

V resnici tudi beležnica počne nekaj podobnega. Avtorji beležnice niso napisali programa, ki "razume" naše ukaze, kot je bil `float(input("Temperatura:"))` in tako naprej. Ne, beležnica (na nek način) pošilja naše programe programu Python in ta jih izvaja. Tisti *na nek način* pa zato, ker je tudi beležnica sama napisana v Pythonu in ga uporabi nekako ... bolj interno, ne prek, bognedaj, ukazne lupine ali česa podobnega.

## Razvojna okolja

Namen gornjega je bil predvsem povedati, da program v Pythonu ni nič drugega kot datoteka z običajnim besedilom in da program `python`, ki izvaja programe v jeziku Python, ne dela drugega, kot da prebere takšno datoteko in izvede navodila, napisana v njej.

Pisati programe z Notepadom bi bilo duhamorno. Notepad nam ne bi pomagal ne pri pisanju ne pri iskanju napak, saj nima pojma o Pythonu. Poleg tega bi morali programe stalno shranjevati in jih poganjati iz ukazne lupine.

Za programiranje uporabljamo razvojna okolja. Razvojna okolja so programi, ki so v osnovi urejevalniki besedil, kot Notepad, vendar nam pomagajo pri delu tako, da smiselno obarvajo kodo, da je preglednejša, dopolnjujejo imena spremenljivk in funkcij, sami dodajo kak delček programa, sproti opozarjajo na morebitne napake, poleg tega pa lahko program, ki ga pišemo, poženemo kar iz razvojnega okolja.

Zadnje sicer ni čisto res. Razvojna okolja (vsaj za Python) tipično ne izvajajo programa sama, temveč pokličejo program Python - glej prejšnji razdelek :).

Pri tem predmetu bomo kot razvojno okolje uporabljali PyCharm, kdor želi, pa lahko uporablja tudi preprostejše urejevalnike.

**Preprosti urejevalniki** Kdor želi kaj preprostega, bo vzel Visual Studio Code. Če namestimo še primerne dodatke, nam pomaga oblikovati kodo, iskati pozabljena imena spremenljivk v dolgih programih, pokazati dokumentacijo posameznih funkcij in, do neke mere, poganjati programe. Sam v njem pišem kratke programčke v Pythonu, za resno programiranje pa uporabljam PyCharm, tako da vam pri morebitnih zapletih z Visual Studio Code ne bom v veliko pomoč.

Starejši mački prisegajo na starejše urejevalnike, kot so emacs in vi oz. vim. Tudi med študenti se vsako leto najde kakšen in nihče mu ne brani. Tudi mazohizem je v sodobni družbi popolnoma sprejeta orientacija.

**Profesionalna okolja** Najbrž najbolj popularno profesionalno okolje za razvoj programov v Pythonu je PyCharm. Gre za komercialno orodje, vendar ga razvija podjetje, ki je zelo naklonjeno tudi odprti kodi, predvsem pa imajo zastonsko različico, ki ji prav nič ne manjka. Zato izjemoma nimamo slabe vesti, ker študente navajamo na komercialne programe. Predvsem pa je PyCharm je v resnici neverjetno dober, prav tako pa so izjemna tudi druga orodja tega podjetja. (Vsem, ki bi se radi naučili vročega novega jezika priporočam Kotlin.)

PyCharm nam bo pri programiranju pomagal še veliko bolj kot Visual Studio Code. Znal bo dopolnjevati napol napisana imena (nekaj malega zna tudi VS Code, a PyCharm je veliko zvitejši), pomagal pri tipih spremenljivk, argumentih funkcij, iskanju napak...

Odpremo torej PyCharm in naredimo projekt "Programiranje 1". Na levi strani vidimo nekakšne direktorije. Naredimo poddirektorij z zaporedno številko in naslovom predavanj in v njem novo datoteko z imenom temperature.py. (Kako to naredimo, boste že odkrili sami, saj niste prvič za računalnikom.) Poklikamo novo datoteko in vanjo vnesemo program.

Ko je program vtipkan, pritisnemo Ctrl-Shift-F10 v Windowsih ali Ctrl-Shift-R na Os X ali bogve kaj v Linuxu (ali pa v meniju Run izberemo Run, vendar se raje navadite na bližnjice, saj boste tole letos naredili deset tisočkrat) in PyCharm bo poglal naš program. Vpis in rezultati bodo v spodnjem delu okna.