Pregled Sintakse

Program se izvaja **statement** (vrstico) po **statement**

(vrstico).

Več enako zamaknjenih vrstic (tab al pa space, je pomembno) skupaj tvorijo blok (block) kode.

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

```
Hello
World!
```

Spremenljivke

Med pisanjem programa bomo ustvarili veliko količino entitet - to so spremenljivke, ki shranjujejo naše vrednosti, kot so string, integer, list, funkcija, class. Te entitete morajo imeti imena, katera so enoznačno (unique) poimenovana.

```
x = 2
```

- x -> ime naše spremenljivke
- = -> znak, ki pomeni, ovrednoti vrednost na desni strani in jo shrani pod ime na

```
# Spremenljivke poimenujemo na sledeč način
x = 2
# x -> enoznačno ime naše spremenljivke
# = -> znak, ki pomeni, ovrednosti vrednost na desni strani in shrani
pod ime na levi strani
# 2 -> vrednost katero želimo shraniti

x = 2# deklariranje integer
x = 2.2# deklarirane float (decimalne številke)
x = True# deklariranje boolean
x = "hello"# deklariranje string
x = ["pingvin", "medved", "los", "volk"] # deklariranje lista
x = ("pingvin", "medved", "los", "volk") # deklariranje toupla
x = {
'macek' : 'Silvestre',
'pes'   : 'Fido',
'papagaj': 'Kakadu'
}   # deklariranje dictionary
x = {1, 2, 3, 4} # deklariranje set
```

String Formating

S prihodom Python3.6 verzije se stringe izpisuje s pomočjo

f-string

```
f'Besedilo {spremenljivka1:format1}, besedilo
naprej{spremenljivka2:format2}, besedilo naprej....'

ime = "Anže"
starost = 10
print(f'{ime:-^10} je {starost:*>10.3f}
{starost*12:e} mesecev.')
# {ime:-^10} ime -> spremenljivka, "-" -> znak s katerim zapolni
mesta, "^" -> naj bo sredinska poravnava, 10 -> 10 znakov
# {starost:*>10.3f} starost -> ime spremenljivke, "*" -> znak s
katerim zapolni mesta, ">" -> poravnava desno, "10" -> 10 znakov za
zapis, ".3" -> naj ima 3 decimalna mesta, "f" -> naj bo to float #
{starost*12:e} starost*12 -> spremenljivka ki jo želimo izpisat, "e"->
naj bo stvar izpisana kot eksponenet (100 -> 1.00e2 ->)
```

```
---Anže--- je ****10.000 let star, oziroma 1.200000e+02 mesecev.
```

Matematične operacije.

-
-
-
-
-
-
-

```
x = 5

y = 3

z = x + y
# ovrednoti, kar je na desni in shra
```

Primerjalne operacije

-
-
-
-
-
-

Naše vrednosti lahko ovrednotimo v boolean kontekstu (ali so True ali False).

```
5 < 10
# v tem primeru smo samo ovrednotili.

# Če želimo lahko to vrednost nato še shranimo.
```

```
x = 5<10

print(x)
```

```
True
```

Logične operacije

- not
- or
- and
- is > Primerja identiteto
- in > Preverja, če je vrednost znotraj primerjalne vrednosti

```
x = False
not x # obrne vrednost. Če je vrednost True jo obrne v False, če je False jo obrne v True
```

```
True
```

```
x = True
y = False
x or y # če je ena izmed vrednosti True, bo izraz True
```

```
True
```

```
x = True
y = False
x and y # če je ena izmed vrednosti False, bo izraz False
```

```
False
```

```
x = 1
x is1# primerja istost, če kažeta na isto mesto
```

```
True
```

```
x = "b"
x in"abc"# primerja ali je x v stringu, listu, itd..
```

```
True
```

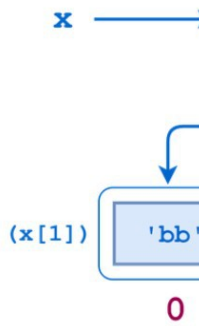
List

Do elementov v list-u lahko dostopamo, če vemo njegov index (na kateri poziciji je).

Na tak način dostopamo do elementov v listi.

```
a  = ["pingvin", "medved"
print(a[0])
print(a[2])
print(a[3])

pingvin
los
volk
```



V Pythonu se indexiranje začne z 0.

```
list[index_elementa]
```

Indexiramo lahko tudi z negativnimi vrednostmi:


```
a  = ["pingvin"
print(a[-5])
print(a[-2])
print(a[-3])

pingvin
volk
los
```

Slicing

T

o

n

a

m

p

o

m

a

g

a

p

r

i

d

o

b

i

t

i

d

o

l

o

č

e

n

e

p

o

d

-

l

i

s

t

e

i

z

ž

e

n

a

r

e

j

e

n

e

l

i

s

-

e

.

```
a[start:konec:korak]

a  = ["pingvin"
print(a[2:5])
print(a[-5:-2])

['los', 'volk', 'medved']
['pingvin', 'medved', 'los']
```

Gnezdenje (nesting)

K

o

t

e

l

e

m

e

n

t

l

i

s

t

a

l

a

h

k

o

s

h

r

a

n

i

š

p

oljubnovrednost.

Tudšeeneedo datenlist.

Dictionary

x

(x[1])

Dostopanje do vrednosti v knjižnici (dictionary)

```
a = ['a', ['bb'
print("A = ", a)

b = a[1]
print("B = ", b)

c = b[1]
print("C = ", c)

d = c[1]
print("D = ", d)

print("Če vse združimo = "
```

```
A =  ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
B =  ['bb', ['ccc', 'ddd'], 'ee', 'ff']
C =  ['ccc', 'ddd']
D =  ddd
Če vse združimo =  ddd
```

Vrednosti najdemo preko ključev.

```
dictionary = {
    ključ: vrednost,
    ključ2: vrednost2
}
```

```
Da dostopamo do vrednosti:
dictionary[ključ]

d = {1: 'a'
print(d)
print(d[2.3
print(d[(1,

{1: 'a', 2.3: 'b', 'string': 'c', (1, 'touple'): 'd'}
```

Izbirni stavki

If stavek

```
if <expr>:
    <statement>
    <statement>
    ...
    <statement>
<following_statement>


<expr> je izraz ovrednoten v Boolean kontekstu,
<statement> je Python izraz (nadaljevanje naše kode), ki je pravilno zamaknjen.
Če je <expr>
potem se  <statement> preskoči
Nato se program nadaljuje z <following_statement>
```

Indentation / Zamikanje

Pri Pythonu se zamikanje (indentation) uporablja za definiranje blokov kode. Vse vrstice z istim zamikom se smatrajo kot isti blok kode. Bloke kode se lahko poljubno globoko "gnezdi". Zamikanje je določeno z tabulatorjem ali presledki. Ni važno točno število, važno je, da je skozi kodo enako.

```
x = 0

y = 5

if x < y:
    print('
    print('
```

kod
e.

```
Smo znotraj if.
End if
End
```

jo
dosežem
o z else.

El se in eli f

```
if <expr>:
    <statement(s)>

else:
    <statement(s)>

Če je <expr>
False se ta blok kode preskoči

x = 100

if x < 50:
    print(
    print(
else:
    print(

    print(

print("Konec"

(drugi blok)

x je velik
Konec

Če želimo še večjo razvejanost naših možnosti lahko uporabimo
(elseif).

if <expr>
    <stat

elif <exp
    <stat

elif <exp
    <stat

else:
```

Vča
sih
želi
mo,
da
če
je
nek
aj
res
se
izve
de
dolo
čen
blok
kod
e,
če
stva
r ni
res
pa
naj
se
izve
de
drug
i del

```
x = 20
if x > 100
    print
elif x >
    print
elif x >
    print
elif x >
    print
else:
    print
print("End"
```

```
x večje

End
```

Loop

While

```
while <expr>:
    <statement>
    <statement>
    ...
    <statement>
<following_statement>
```

Če je <expr> ovrednoten izvedbu tega block-a, se ponovno vrnemo na <expr> ovrednotimo.

```
i = 0
while i <
   print(
    i +=
```

```
Ponovljeno 0x krat Ponovljeno 1x krat Ponovljeno 2x krat Ponovljeno 3x krat Ponovljeno 4x krat Ponovljeno 5x krat Ponovljeno 6x krat Ponovljeno 7x krat Ponovljeno 8x krat Ponovljeno 9x krat
```

For

Primer: kadar hočemo izvesti blok kode za vsak element v list-u.

```
for <element>
    <statement>
    <statement>
    <statement>
    ...
<following_statement>

primes = [
for prime
    print
```

```
2 is  a prime number.  3 is  a prime number. 5 is  a prime number. 7 is a prime number. 11 is  a prime number.
```

Break

Break keyword terminira najbolj notranjo zanko v kateri se nahaja.

```
avti = [

for avto
    if avto ==
    print
    break
    print
print("End"
```

```
Avto je ok.
Avto je ok.
Avto je ok.
Avto je zanič.
End
```

Continue

Continue keyword izpusti kodo, ki se more še izvesti, in skoči na naslednjo iteracijo zanke.

```
avti = [

for avto
    if avto ==
    print
    continue
    elif
```

```
    print
    print
print("Konec"

Avto je ok.

Avto je ok.

Avto je ok.

Avto je zanič.
Avto je ok.

End

# Primer: Filtriranje elementov
a = [1,

even_squares = [x**
print(a)
print(even_squares)

[1, 2,

[36]
```

Funkcije

```
def  pozdrav(ime):
    print
    return

# funkcijo kličemo z uporabo njenega imena in (). Znotraj () po
potrebne argumente, če jih funkcija zahteva
x = pozdrav(
print(x)

Živjo, Jaka. Good morning!
Jaka
```

Razred/Class

```
* razred/
naslednje vrstice tvorijo
* Pes -> ime našega razreda
* : -> označuje konec definicije razreda
* vrsta =
```

```
instancam razreda)
* hrana = [
(vsem instancam razreda)
* def__init__
ustvarjanju nove instance razreda
* def opis(
instancam razreda)

class Pes:
    vrsta =
    hrana = [
   #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def




    def


    def
    self
Class variable. Od kle naprej ni važn, če spreminjaš "Pes.vrsta =
xxx", ta instanca bo ohranla svojo vrednost

    def
    self
mutable to vpliva na vse instance razreda

fido = Pes(
print(fido.ime)
print(fido.opis())
print(Pes.vrsta)

Fido
Fido je star 9
pes
```