In [ ]:

```
```

Vaja 01

Iz sledečega list-a pridobite vrednost **ffff**

```
our_list = ["a", ["bb", "cc"], "d", [["eee"], ["ffff"], "ggg"]]
```

In [ ]:

```
our_list = ["a", ["bb", "cc"], "d", [["eee"], ["ffff"], "ggg"]]

print(our_list)
print(our_list[3])
print(our_list[3][1])
print(our_list[3][1][0])
```

Vaja 02

Pri sledečem list-u začnite z vrednostjo 4 in vzemite vsako 3 vrednost. Dobljene številke shranite v nov touple.

```
our_list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

Rešitev:
(4, 7, 10, 13, 16, 19)
```

In [1]:

```
our_list = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

our_tuple = tuple(our_list[3::3])
print(type(our_tuple))
print(our_tuple)
```

```
<class 'tuple'>
(4, 7, 10, 13, 16, 19)
```

Vaja 03

Sledečemu dictionary zamenjanjte vrednost pod ključem **b** v vrednost 12 in odstranite vrednost pod ključem **d**.

In [2]:

```
our_dict = {
    "a": 10,
    "b": 9,
    "c": 8,
    "d": 7,
    "e": 3
}

our_dict["b"] = 12
del our_dict["d"]

print(our_dict)
```

```
{'a': 10, 'b': 12, 'c': 8, 'e': 3}
```

FLOW CONTROL STATEMENTS

Omogočajo nam kontrolo sprememb in logike programa.

If statement

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

```
<expr> je izraz ovrednoten v Boolean kontekstu.
<statement> je Python izraz (nadaljevanje naše kode), ki je pravilno zamakn
jen.
Če je <expr> True, potem se izvedejo <statement>. Če je <expr> False, potem
se  <statement> preskoči in se ne izvede.
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 "nesta".

> Zamikanje je določeno z tabulatorjem ali presledki. Ni važno točno število, važno je, da je skozi kodo enako.

In [5]:

```
x = 0
y = 5

if x < y:
    print("Smo znotraj if.")
    print("End if")
print("End")
```

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

Else

Včasih želimo, da če je nekaj res se izvede določen blok kode, če stvar ni res pa naj se izvede drug del kode.

To dosežemo z else.

```python if : <statement(s)> else: <statement(s)>

Če je True se izvede blok direktno pod njem, če pa je False se ta blok kode preskoči in se izvede blok pod else.
```

In [4]:

```
x = 100

if x < 50:
    print('(first block)')
    print('x is small')
else:
    print('(second block)')
    print('x is large')

print("End")
```

```
(second block)
x is large
End
```

Elif

```python Če želimo še večjo razvejanost naših možnosti lahko uporabimo elif (else if).

if : <statement(s)> elif : <statement(s)> elif : <statement(s)> else: <statement(s)>

Python preveri vsak posebej. Pri ta prvem, ki bo True, bo izvedel njegov blok kode. Če ni nobeden True se bo izvedel else blok kode. ```
```

In [5]:

```
x = 20
if x > 100:
    print('x je večje od 100')
elif x > 50:
    print('x večje od 50 in manjše od 100')
elif x > 30:
    print('x večje od 30 in manjše od 50')
elif x > 10:
    print('x večje od 10 in manjše od 30')
else:
    print("x manjše od 10")

print("End")
```

```
x večje od 10 in manjše od 30
End
```

One-line if statement

Obstaja način zapisa if stavka v eni vrstici ampak se ta način odsvetuje, ker napravi kodo nepregledno.

```
<expr1> if <conditional_expr> else <expr2>

z = 1 + x if x > y else y + 2

If <conditional_expr> is true, <expr1> is returned and <expr2> is not evalu
ated.
If <conditional_expr> is false, <expr2> is returned and <expr1> is not eval
uated.
```

In [6]:

```
x = 8
z = 1 + x if x > 10 else x**2
print(z)

x = 20
z = 1 + x if x > 10 else x**2
print(z)
```

```
64
21
```

The pass statements

Uporablja se kot "placeholder", da nam interpreter ne meče napak.

In [11]:

```
if True:

print("Hello") # should give IndentationError
```

```
  File "<ipython-input-11-33a91c099307>", line 3
    print("Hello") # should give IndentationError
        ^
IndentationError: expected an indented block
```

In [12]:

```
if True:
    pass
print("Hello") # should be fine now with the pass added
```

```
Hello
```

Vaja 01

Napišite program, ki bo uporabnika uprašal naj vnese neko celoštevilsko vrednost. Program naj nato izpiše ali je vrednost deljiva z 3 ali ne.

In [2]:

```
x = int(input("Vnesi celoštevilsko vrednost: "))
if x%3 == 0:
    print("Število je deljivo s 3")
else:
    print("Število ni deljivo s 3")
```

```
Vnesi celoštevilsko vrednost: 1
Število ni deljivo s 3
```

Vaja 02

Napišite program, ki bo pretvoril stopinje Celzija v Fahrenheit ali obratno.

Uporabnik naj vnese številko. Nato naj vnese v katerih enotah nam je podal vrednost (**C** ali **F**). Glede na vnešeno črko naj vaš program uporabi pravilno formulo za pretvorbo.

$$T(°F) = T(°C) \times 9/5 + 32$$

$$T(°C) = (T(°F) - 32) \times 5/9$$

Če uporabnik ni vnesel **C** ali **F** naj program izpiše *Prišlo je do napake.*

Primer:

```
Vnesi vrednost: 12
Vnesi enoto: C
```

Rešitev:

```
    12 stopinj celzija je enako 53.6 fahrenheit.
```

In [24]:

```
stopinje = float(input("Vnesi vrednost: "))
enota = input("V katerih enotah je podana vrednost? [C/F]: ")

if enota == "C":
    fahrenheit = stopinje*9/5 + 32
    print(f"{stopinje} {enota} je enako {fahrenheit} fahrenheit.")
elif enota == "F":
    celsius = (stopinje - 32)*5/9
    print(f"{stopinje} {enota} je enako {celsius} celsius.")
else:
    print("Prišlo je do napake")
```

```
Vnesi vrednost: -50
V katerih enotah je podana vrednost? [C/F]: C
-50.0 C je enako -58.0 fahrenheit.
```

While

While zanka deluje na podoben princip kot if. While izvaja blok kode, dokler je "expression" True.

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

In [26]:

```
lepo_vreme = True
while lepo_vreme:
  print('Vreme je lepo.')
  lepo_vreme = False
```

```
Vreme je lepo.
```

In [14]:

```
#the body should be able to change the condition’s value, because if the condition
#True at the beginning, the body might run continuously to infinity
#while True:
#    print("Neskončna zanka. Se ne ustavim.")

#ustavimo v CTRL + C
```

While zanko se lahko uporabi za ponovitev bloka kode določenega števila korakov.

In [27]:

```
i = 0
while i < 10:
    print(f'Repeated {i} times')
    i += 1
```

```
Repeated 0 times
Repeated 1 times
Repeated 2 times
Repeated 3 times
Repeated 4 times
Repeated 5 times
Repeated 6 times
Repeated 7 times
Repeated 8 times
Repeated 9 times
```

In [16]:

```
#A common use of the while loop is to do things like these:

temperature = 15

while temperature < 20:
    print('Heating...')
    temperature += 1

#Only instead of the temperature increasing continuously, we would e.g. get it from
#Remember to always have a way of exiting the loop! Otherwise it will run endlessly
```

```
Heating...
Heating...
Heating...
Heating...
Heating...
```

Obstaja tud while else.

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

The <additional_statement(s)> specified in the else clause will be executed when the while loop terminates.
About now, you may be thinking, “How is that useful?” You could accomplish the same thing by putting those statements immediately after the while loop , without the else:

What’s the difference?

In the latter case, without the else clause, <additional_statement(s)> will be executed after the while loop terminates, no matter what.

When <additional_statement(s)> are placed in an else clause, they will be executed only if the loop terminates “by exhaustion”—that is, if the loop iterates until the controlling condition becomes false. If the loop is exited by a break statement, the else clause won’t be executed.

Vaja 01

Napišite program, ki izpiše prvih 10 sodih števil.

In [29]:

```
counter = 0
number = 1

while counter<10:
    if number%2 == 0:
        print(number)
        counter += 1
    number += 1
```

```
2
4
6
8
10
12
14
16
18
20
```

Vaja 02

Uporabnik naj vnese željeno dolžino Fibonaccijevega zaporedja. Program naj nato to zaporedje shrani v list in ga na koncu izpiše.

```
Fibonacci sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

In [3]:

```
x = int(input("Dolžina Fibonnacijevega zaporedja: "))
fibonacci = [0, 1]
counter = 2

while counter < x: # while len(fibonacci) < x  bi tud šlo
    fibonacci.append(fibonacci[-1] + fibonacci[-2])
    counter += 1
print(fibonacci)
```

```
Dolžina Fibonnacijevega zaporedja: 10
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

For loop

Uporablja se kadar hočemo izvesti blok kode za vnaprej določeno število ponovitev.

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

```
for <var> in <iterable>:
    <statement(s)>
```

In [17]:

```
primes = [2, 3, 5, 7, 11] #itrable
for prime in primes:
    print(f'{prime} is  a prime number.')
```

```
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.
```

In [18]:

```
kid_ages = (3, 7, 12)
for age in kid_ages:
    print(f'I have a {age} year old kid.')
```

```
I have a 3 year old kid.
I have a 7 year old kid.
I have a 12 year old kid.
```

Velikokrat se skupaj z for-loop uporablja funkcija range().

```
range(start, stop, step)
```

- start - Optional. An integer number specifying at which position to start. Default is 0
- stop - An integer number specifying at which position to end, excluding this number.
- step - Optional. An integer number specifying the incrementation. Default is 1

Funkcija range nam zgenerira list števil.

In [24]:

```
x = range(-5, 10, 1)
print(type(x))
print(list(x))
```

```
<class 'range'>
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [41]:

```
# Primer: Iteracija čez dictionary
pets = {
    'macka': 6,
    'pes': 12,
    'krava': 20
}

for pet, years in pets.items():
    print(f'{pet} je star/a {years} let.')
```

```
macka je star/a 6 let.
pes je star/a 12 let.
krava je star/a 20 let.
```

> **Nasveti**

- Use the enumerate function in loops instead of creating an “index” variable

In [ ]:

```

```

Programmers coming from other languages are used to explicitly declaring a variable to track the index of a container in a loop. For example, in C++:

```
for (int i=0; i < container.size(); ++i)
{
// Do stuff
}
```

In Python, the enumerate built-in function handles this role.

In [7]:

```
moj_list = ["Anže", "Luka", "Mojca"]
index = 0
for element in moj_list:
    print (f'{index} {element}')
    index += 1
```

```
0 Anže
1 Luka
2 Mojca
```

In [6]:

```
#Idiomatic
moj_list = ["Anže", "Luka", "Mojca"]
for index, element in enumerate(moj_list):
    print (f'{index} {element}')
```

```
0 Anže
1 Luka
2 Mojca
```

Break

Break keyword terminira najbolj notranjo zanko v kateri se nahaja.

In [44]:

```
avti = ["ok", "ok", "ok", "slab", "ok"]

for avto in avti:
    if avto == "slab":
        print("Avto je zanič.")
        break
    print("Avto je ok.")
    print("Naslednji korak zanke")
print("End")
```

```
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je zanič.
End
```

Continue

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

In [45]:

```
avti = ["ok", "ok", "ok", "slab", "ok"]

for avto in avti:
    if avto == "slab":
        print("Avto je zanič.")
        continue #continue #lah pokažeš še primer k je stvar zakomentirana
    print("Avto je ok.")
    print("Naslednji korak zanke")
print("End")
```

```
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je ok.
Naslednji korak zanke
Avto je zanič.
Avto je ok.
Naslednji korak zanke
End
```

In [ ]:

```

```

Vaja 01

Iz danega dictionary izpišite vse ključe, katerih vrednost vsebuje črko **r**.

In [81]:

```
d = {
    "mačka": "Micka",
    "pes": "Fido",
    "volk": "Rex",
    "medved": "Žan",
    "slon": "Jan",
    "žirafa": "Helga",
    "lev": "Gašper",
    "tiger": "Anže",
    "papagaj": "Črt",
    "ribica": "Elena",
    "krokodil": "Kasper",
    "zajec": "Lars",
    "kamela": "Manca"
}


for key,value in d.items():
    if "r" in value or "R" in value:
        print(f"{key}")
```

```
volk
lev
papagaj
krokodil
zajec
```

In [ ]:

```

```

In [ ]:

```

```

Vaja 02

Poiščite vsa praštevila med 2 in 30.

In [2]:

```
for num in range(2,31):
    prime = True
    for i in range(2,num):
        #print(f"{num} / {i}. Ostanek je {num%i}")
        if (num%i==0):
            prime = False
            break
    if prime:
        print(f"{num} JE praštevilo!")
    #else:
        #print(f"{num} NI praštevilo.")
```

```
2 JE praštevilo!
3 JE praštevilo!
5 JE praštevilo!
7 JE praštevilo!
11 JE praštevilo!
13 JE praštevilo!
17 JE praštevilo!
19 JE praštevilo!
23 JE praštevilo!
29 JE praštevilo!
```

Funkcije

Funkcija je blok kode, ki izvede specifično operacijo in jo lahko večkrat uporabimo.

> Za primer, če v programu večkrat uporabniku rečemo, naj vnese celo število med 1 in 20. Od njega zahtevamo vnos s pomočjo **input** in nato to spremenimo v celo število z uporabo **int**. Nato preverimo ali je število v pravilnem rangu. To zaporedje kode v programu večkrat ponovimo.

> Če se sedaj odločimo, da naj uporabnik vnese celo število v rangu med 1 in 100, moramo popraviti vsako vrstico posebej, kar hitro lahko privede do napake.

> Za lažje pisanje programa lahko to zaporedje kode shranimo v funkcijo. Če sedaj spremenimo rang, le-tega popravimo samo enkrat, znotraj naše funkcije.

Funkcije nam omogočajo uporabo tuje kode brez globjega razumevanja kako le-ta deluje. Z njihovo pomočjo lahko zelo kompleksne probleme razbijemo na majhne in bolj obvladljive komponente.

Defining a Function

Funkcijo definiramo z uporabo def keyword kateri sledi ime funkcije in navadni oklepaji (). Zaključi se jo z " : ".

Blok kode, katero želimo, da naša funkcija izvede zapišemo z ustreznim zamikom.

```
def ime_funkcije():
    # Naš blok kode katero želimo izvesti
    x = input("...")
    y = int(x) + 5
    ...
```

Po priporočilih se imena funkcije piše na snake_case način (vse male črke, med besedami podčrtaj _)

Funkcijo nato uporabimo tako, da jo pokličemo po imenu in dodamo zraven ().

```
ime_funkcije() # Klic naše funkcije
```

In [28]:

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

print("Začetek programa")
hello()
print("Nadaljevanje programa")
#pokažemo, da moremo funkcijo klicat po definiciji.
#pazt, če to kažeš v jupyter notebooku, k tm se shranjo stvari v ozadju
```

```
Začetek programa
Hello, World!
Nadaljevanje programa
```

Funkcije je v kodi potrebno ustvariti, še predno jo kličemo.

In [30]:

```
print("Začetek programa")
hello2()
print("Nadaljevanje programa")

def hello2():
    print("Hello, World!")
```

```
Začetek programa

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
 last)
<ipython-input-30-d0c6cd4e0154> in <module>
      1 print("Začetek programa")
----> 2 hello2()
      3 print("Nadaljevanje programa")
      4
      5 def hello2():

NameError: name 'hello2' is not defined
```

In [ ]:

Naloga:

Napišite funkcijo, ki od uporabnika zahteva naj vnese svojo EMŠO število.

Funkcija naj nato izpiše koliko let je uporabnik star.

EMŠO ima 14 številk XXXXyyyXXXXXXX. 5.,6.,7. številka predstavljajo letnico rojstva (999 -> 1999 leto rojstva).

Primeri:

```
Input:
Vnesi emšo: 0102999500111

Output:
Star si 22 let

Input:
Vnesi emšo: 0104986505555

Output:
Star si 35 let
```

In [6]:

```
# Rešitev
def fun():
    emšo = input("Vnesi emšo: ")

    letnica = int(emšo[4:7]) + 1000
    print(f"Star si {2021-letnica} let")

fun()
```

```
Vnesi emšo: 0104986505555
Star si 35 let
```

Working with Parameters

Funkciji lahko pošljemo določene spremenljivke, katere želimo uporabiti v funkciji.

> Primer: Če vemo ime uporabnika, ga lahko kličemo po imenu, kadar od njega zahtevamo input.

Vrednost, ki jo pošljemo v funkcijo, se reče **argument**. To funkcija sprejme kot **parameter**.

- Parameters are the name within the function definition.
- Arguments are the values passed in when the function is called.

Parametre funkcije definiramo znotraj njenih "( )".

```
def funkcija_1(x, y, z): # x, y, z are parameters
    pass

funkcija_1(1, 2, 3) # 1, 2, 3 are arguments
```

In [9]:

```
def funkcija_1(x, y, z):
    print(f"X vrednost: {x}")
    print(f"Y vrednost: {y}")
    print(f"Z vrednost: {z}")

funkcija_1(1,2,3)
```

```
X vrednost: 1
Y vrednost: 2
Z vrednost: 3
```

V zgornjem primeru se ob klicu funkcije:

- vrednost 1 shrani v spremenljivko x
- vrednost 2 shrani v spremenljivko y
- vrednost 3 shrani v spremenljivko z

Zato je vrstni red argumentov pomemben!

In [10]:

```
def funkcija_1(x, y, z):
    print(f"X vrednost: {x}")
    print(f"Y vrednost: {y}")
    print(f"Z vrednost: {z}")

funkcija_1(1, 2, 3)
print("Zamenjajmo vrstni red.")
funkcija_1(3, 2, 1)
```

```
X vrednost: 1
Y vrednost: 2
Z vrednost: 3
Zamenjajmo vrstni red.
X vrednost: 3
Y vrednost: 2
Z vrednost: 1
```

Pomembno je tudi, da podamo pravilno število argumentov!

Če funkcija pričakuje 3 argumente, ji moramo podatki 3 argumente. Nič več. nič manj. V nasprotnem primeru dobimo napako.

In [18]:

```
# Primer, ko podamo premalo argumentov
def funkcija_1(x, y, z):
    print(f"X vrednost: {x}")
    print(f"Y vrednost: {y}")
    print(f"Z vrednost: {z}")

funkcija_1(1, 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
 last)
<ipython-input-18-e9b6b54ff80a> in <module>
      4     print(f"Z vrednost: {z}")
      5
----> 6 funkcija_1(1, 2)

TypeError: funkcija_1() missing 1 required positional argument: 'z'
```

In [19]:

```
# Primer, ko podamo preveč argumentov
def funkcija_1(x, y, z):
    print(f"X vrednost: {x}")
    print(f"Y vrednost: {y}")
    print(f"Z vrednost: {z}")

funkcija_1(1, 2, 3, 4)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
 last)
<ipython-input-19-271e80339153> in <module>
      4     print(f"Z vrednost: {z}")
      5
----> 6 funkcija_1(1, 2, 3, 4)

TypeError: funkcija_1() takes 3 positional arguments but 4 were given
```

In [ ]:

Naloga:

Napiši funkcijo, ki sprejme 3 argumente.

Funkcija naj izpiše kateri ima največjo vrednost in koliko je ta vrednost.

Primeri:

```
Input:
fun_01(0,-5,6)

Output:
Tretji argument je največji. Vrednost: 6

Input:
fun_01(1, 50, -50)

Output:
Drugi argument je največji. Vrednost: 50
```

In [12]:

```
# Rešitev
def fun_01(a, b, c):
    if a>=b and a>=c:
        print(f"Prvi argument je največji. Vrednost: {a}")
    if b>=a and b>=c:
        print(f"Drugi argument je največji. Vrednost: {b}")
    if c>=b and c>=b:
        print(f"Tretji argument je največji. Vrednost: {c}")


fun_01(0,-5,6)
fun_01(1, 50, -50)
```

```
Tretji argument je največji. Vrednost: 6
Drugi argument je največji. Vrednost: 50
```

Keyword Arguments

Naše argumente lahko poimenujemo s pravilnim imenom parametra in tako, ko naslednjič kličemo funkcijo, ne potrebujemo argumente podati v pravilnem vrstnem redu.

```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav(priimek="Novak", naslavljanje="gospod", ime="Miha")
```

In [46]:

```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav("gospod", "Miha", "Novak")
print("\nUporaba Keyword arguments\n")
pozdrav(priimek="Novak", naslavljanje="gospod", ime="Miha")
```

```
Pozdravljeni gospod Miha Novak.

Uporaba Keyword arguments

Pozdravljeni gospod Miha Novak.
```

Če podamo napačno ime, dobimo napako.

In [47]:

```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav(zadnje_ime="Novak", naslavljanje="gospod", ime="Miha")
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-47-89652f3d516a> in <module>
      2     print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")
      3
----> 4 pozdrav(zadnje_ime="Novak", naslavljanje="gospod", ime="Miha")

TypeError: pozdrav() got an unexpected keyword argument 'zadnje_ime'
```

Pri klicanju funkcije lahko uporabimo oba načina podajanja argumentov. Vendar je pomemben vrstni red.

In [48]:

```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav("gospod", "Miha", priimek="Novak")
```

```
Pozdravljeni gospod Miha Novak.
```

In [49]:

```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav("gospod", priimek="Novak", "Miha")
```

```
  File "<ipython-input-49-d1b39220fd0c>", line 4
    pozdrav("gospod", priimek="Novak", "Miha")
                                       ^
SyntaxError: positional argument follows keyword argument
```

Default Argument Values

Za naše parametre lahko določimo default vrednost, v primeru, da ob klicu funkcije argumenta ne podamo.

```
def funkcija(x=1, y=2):
    print(x + y)

funkcija() # Funkcijo kličemo brez argumentov

Output: 3 # Privzeti vrednosti sta x=1 in y=2
```

In [56]:

```
def pozdrav(naslavljanje="gospod", ime="Miha", priimek="Novak"):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

pozdrav()

pozdrav("g.", "Andrej", "Kovač")
pozdrav(ime="Gregor")
```

```
Pozdravljeni gospod Miha Novak.
Pozdravljeni g. Andrej Kovač.
Pozdravljeni gospod Gregor Novak.
```

Potrebno je paziti, da so parametri z default vrednostjo definirani za parametri brez default vrednosti.

In [60]:

```
def funkcija(x, y, z=0):
    print(x + y + z)

funkcija(1, 2)
```

```
3
```

In [62]:

```
def funkcija(x, y=0, z):
    print(x + y + z)

funkcija(1, 2, 3)
```

```
  File "<ipython-input-62-d290ea3a79c4>", line 1
    def funkcija(x, y=0, z):
                         ^
SyntaxError: non-default argument follows default argument
```

In [ ]:

```

```

Naloga:

Napišite funkcijo, ki izpiše prvih N največjih vrednosti v podanem listu.

Funkcija naj ima dva parametra. Prvi parameter je list, znotraj katerega bomo iskali največje vrednosti. Drugi parameter število, ki nam pove koliko prvih največjih števil naj izpišemo. Če vrednost ni podana, naj se izpiše prvih 5 največjih števil.

Primeri:

```
Input:
vaja([1,5,7,-2,3,8,2-5,12,-22])

Output:
12
8
7
5
3

Input:
vaja([1,5,7,-2,3,8,2-5,12,-22], 3)

Output:
12
8
7
```

In [69]:

```
# Rešitev

def vaja(l, n=5):
    for _ in range(n):
        max_ = max(l)
        print(max_)
        l.remove(max_)


vaja([1,5,7,-2,3,8,2-5,12,-22])
print()
vaja([1,5,7,-2,3,8,2-5,12,-22], 3)
```

```
12
8
7
5
3

12
8
7
```

*args and **kwargs

Ta dva parametra nam omogočata, da funkciji pošljemo poljubno število argumentov.

*args nam pove, da naj neznane argumente zapakira v touple imenovan args.

**kwargs nam pove, da naj neznane argumente zapakira v dictionary imenovan kwargs.

> http://book.pythontips.com/en/latest/args_and_kwargs.html (http://book.pythontips.com/en/latest/args_and_kwargs.html)

> The idiom is also useful when maintaining backwards compatibility in an API. If our function accepts arbitrary arguments, we are free to add new arguments in a new version while not breaking existing code using fewer arguments. As long as everything is properly documented, the “actual” parameters of a function are not of much consequence.

> First of all let me tell you that it is not necessary to write *args or **kwargs. Only the * (asterisk) is necessary. You could have also written *var and **vars. Writing *args and **kwargs is just a convention.

In [77]:

```
def test_args(a, b, c, *args):
    print(f"a = \t {a}")
    print(f"b = \t {b}")
    print(f"c = \t {c}")
    print(f"args = \t {args}")

test_args(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
a =      1
b =      2
c =      3
args =   (4, 5, 6, 7, 8, 9)
```

In [75]:

```
# Primer *ARGS

def sestevalnik(*args):
    value = 0
    for ele in args:
        value += ele
    print(value)

sestevalnik(1, 2, 3)

sestevalnik(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
6
45
```

In [ ]:

```

```

In [78]:

```
def test_kwargs(a, b, c, **kwargs):
    print(f"a = \t {a}")
    print(f"b = \t {b}")
    print(f"c = \t {c}")
    print(f"kwargs = \t {kwargs}")


test_kwargs(a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8, i=9)
```

```
a =      1
b =      2
c =      3
kwargs =         {'d': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9}
```

**kwargs pridejo prav pri posodabljanju kode in ohranjanju podpore za starejše verzije kode.

Primer: ustvarimo funkcijo **moja_funkcija**, ki ima parameter *barva_grafa*. Drugi programerij uporabijo mojo funkcijo.

Kasneje se odločim posodobiti mojo funkcijo tako, da spremenim ime parametra v *barva*. Sedaj bi morali vsi drugi programerij, ki so uporabili mojo funkcijo prav tako posodobiti njihovo kodo. Z uporabo **kwargs pa lahko še vedno zajamemo njihove argumente.

In [86]:

```
def moja_funkcija(podatki, barva_grafa="črna"):
    print(f"Barva grafa je {barva_grafa}.")

moja_funkcija([1,2,3], barva_grafa="rdeča")
```

```
Barva grafa je rdeča.
```

In [87]:

```
# Želi se posodobit to funcijo
def moja_funkcija(podatki, barva="črna"):
    print(f"Barva grafa je {barva}.")

moja_funkcija([1,2,3], barva_grafa="rdeča")
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-87-8300209fd37d> in <module>
      3     print(f"Barva grafa je {barva_grafa}.")
      4
----> 5 moja_funkcija([1,2,3], barva_grafa="rdeča")

TypeError: moja_funkcija() got an unexpected keyword argument 'barva_g
rafa'
```

In [89]:

```
# Želi se posodobit to funcijo
def moja_funkcija(podatki, barva="črna", **kwargs):
    if "barva_grafa" in kwargs.keys():
        print(f"Barva grafa je {kwargs['barva_grafa']}.")
    else:
        print(f"Barva grafa je {barva}.")

moja_funkcija([1,2,3], barva_grafa="rdeča")
```

```
Barva grafa je rdeča.
```

In [ ]:

```

```

In [ ]:

```

```

Returning a Value

Vsaka funkcija tudi vrne določeno vrednost.

Če funkciji nismo eksplicitno določili katero vrednost naj vrne, vrne vrednost **None**.

In [90]:

```
def funkcija():
    print("Pozdrav")

x = funkcija()
print(x)
```

```
Pozdrav
None
```

Da vrnemo specifično vrednost uporabimo besedo **return**.

```
def sestevalnik(x, y):
    vsota = x + y
    return vsota

x = sestevalnik(1, 2)
print(x)

Output: 3
```

In [93]:

```
def sestevalnik(x, y):
    print("Seštevam...")
    vsota = x + y
    return vsota

x = sestevalnik(1, 2)
print(x)
```

```
Seštevam...
3
```

Ko se izvede ukaz **return** se vrne vrednost in koda znotraj funkcije se neha izvajati.

In [94]:

```
def sestevalnik(x, y):
    print("Seštevam...")
    vsota = x + y
    return vsota
    print("Končano")

x = sestevalnik(1, 2)
print(x)
```

```
Seštevam...
3
```

Znotraj funkcije imamo lahko tudi več **return** statements, ki vrnejo različne vrednosti, glede na logiko funkcije.

In [98]:

```
def vecje_od_5(x):
    if x > 5:
        return True
    elif x <= 5:
        return False

print(vecje_od_5(1))
print(vecje_od_5(10))
```

```
False
True
```

In [ ]:

```

```

Returning Multiple Values

Funkcija lahko vrne le eno vrednost (bolje rečeno: le en objekt).

Če želimo vrniti več vrednosti jih preprosto zapakiramo v list, touple, dictionary in posredujemo tega.

In [100]:

```
def add_numbers(x, y, z):
    a = x + y
    b = x + z
    c = y + z
    return a, b, c # isto kot return (a, b, c)

sums = add_numbers(1, 2, 3)
print(sums)
print(type(sums))
```

```
(3, 4, 5)
<class 'tuple'>
```

In [ ]:

```

```

In [ ]:

```

```

Naloga:

Napišite funkcijo, ki sprejme nabor podatkov v obliki dictionary in vrne največjo vrednost vsakega ključa.

Primeri:

```
Input:
data = {"prices": [41970, 40721, 41197, 41137, 43033],
       "volume": [49135346712, 50768369805, 47472016405, 34809039137, 38700
661463]}
funkcija(data)

Output:
[43033, 50768369805]
```

In [110]:

```
data = {"prices": [41970, 40721, 41197, 41137, 43033],
       "volume": [49135346712, 50768369805, 47472016405, 34809039137, 38700661463]}

def funkcija(data):
    r = []
    for key, value in data.items():
        #print(key, value)
        r.append(max(value))
    return r

print(funkcija(data))
```

```
[43033, 50768369805]
```

Zanimivosti

Python funkcije so objekti. Lahko jih shranimo v spremenljivke, lahko jih posredujemo kot argumente ali vrnemo kot vrednost funkcije.

In [100]:

```
def hello(name):
    return f'My name is {name}'
```

In [101]:

```
print(hello("Gregor"))
```

```
My name is Gregor
```

In [102]:

```
funkcija = hello
print(funkcija("Gregor"))
print(funkcija)
print(type(funkcija))
```

```
My name is Gregor
<function hello at 0x0000015411EE6A60>
<class 'function'>
```

In [103]:

```
func = [hello, 2 ,3, 'Janez']
print(func[0](func[3]))
```

```
My name is Janez
```

In [ ]:

```

```

Naloga:

Ustvarite funkcijo, ki kot parametra vzeme list številk in neko število **m**, ki predstavlja zgornjo mejo.

Funkcija naj se sprehodi skozi podan list in vsako število, ki je večje od m, spremeni v m.

Funkcija naj na koncu vrne spremenjen list.

Primeri:

```
Input:
funkcija([1,12,-3,54,12,-22,65,32], 33)

Output:
[1, 12, -3, 33, 12, -22, 33, 32]
```

In [117]:

```
# Rešitev
def funkcija(l, m):
    new_l = []
    for ele in l:
        if ele > m:
            new_l.append(m)
        else:
            new_l.append(ele)

    return new_l

print(funkcija([1,12,-3,54,12,-22,65,32], 33))
```

```
[1, 12, -3, 33, 12, -22, 33, 32]
```

In [ ]:

```

```

Naloga:

Ustvari funkcijo, ki uredi list po vrstnem redu. Sprejme naj list in ukaz **asc** (naraščajoči vrstni red) ali **desc** (padajoči vrstni red). List naj nato ustrezno uredi. V kolikor ukaz ni posredovan naj bo default vrednost **asc**.

Primeri:

```
Input:
fun_03([1,4,2,8,4,0], ukaz="desc")

Output:
[8, 4, 4, 2, 1, 0]


Input:
fun_03([1,4,2,8,4,0], ukaz="asc")

Output:
[0, 1, 2, 4, 4, 8]


Input:
fun_03([5,8,-2,13,6,-6])

Output:
[-6, -2, 5, 6, 8, 13]
```

In [115]:

```
def fun_03(old_list, ukaz="asc"):
    new_list = []
    if ukaz == "asc":
        while old_list:
            minimum = old_list[0]
            for i in old_list:
                if i < minimum:
                    minimum = i
            new_list.append(minimum)
            old_list.remove(minimum)
    if ukaz == "desc":
        while old_list:
            maximum = old_list[0]
            for i in old_list:
                if i > maximum:
                    maximum = i
            new_list.append(maximum)
            old_list.remove(maximum)

    return new_list

print(fun_03([1,4,2,8,4,0], ukaz="desc"))
print(fun_03([1,4,2,8,4,0], ukaz="asc"))
print(fun_03([5,8,-2,13,6,-6]))
```

```
[8, 4, 4, 2, 1, 0]
[0, 1, 2, 4, 4, 8]
[-6, -2, 5, 6, 8, 13]
```

In [ ]:

In [ ]:

In [ ]:

Lambda funkcija

Lambda funkcije so anonimne funkcije, kar pomeni, da nimajo imena (niso vezane na spremenljivko).

> Anonimna funkcija - anonymous function is a function that is defined without a name.

> We have already seen that when we want to use a number or a string in our program we can either write it as a literal in the place where we want to use it or use a variable that we have already defined in our code. For example, print("Hello!") prints the literal string "Hello!", which we haven't stored in a variable anywhere, but print(message) prints whatever string is stored in the variable message.

> We have also seen that we can store a function in a variable, just like any other object, by referring to it by its name (but not calling it). Is there such a thing as a function literal? Can we define a function on the fly when we want to pass it as a parameter or assign it to a variable, just like we did with the string "Hello!"?

> A lambda function may only contain a single expression, and the result of evaluating this expression is implicitly returned from the function (we don’t use the return keyword)

```
lambda x,y : x +y
```

Sestavljene so iz:

- lambda - keyword
- parametri so napisani med lambda in :
- "single expression" (1 vrstica kode). Rezultat / vrednost tega "single expression" se vrne kot vrednost funkcije

In [1]:

```
(lambda x, y: x+y)(2, 3)
```

Out[1]:

```
5
```

In [2]:

```
add = lambda x, y: x + y
print(add)
print(type(add))
```

```
<function <lambda> at 0x000001D590FDDE50>
<class 'function'>
```

In [5]:

```
add(5,3)
```

Out[5]:

```
8
```

Primer, če bi zgornjo lambda funkcijo napisalo kot navadno funkcijo.

In [6]:

```
def add(x, y):
    return x + y
```

In [ ]:

```

```

Lambda funkcije pridejo najbolj do izraza, kjer je treba kot argument posredovati funkcijo. Namesto dejanske funkcije lahko posredujemo lambda funkcijo.

Za primer vzemimo funkcijo `sorted()`.

https://docs.python.org/3/library/functions.html#sorted (https://docs.python.org/3/library/functions.html#sorted)

Naša naloga je sortirati sledeče vrednosti glede na **market_cap** vrednost, od največje do najmanjše.

In [13]:

```
data = [
  {
    "id": "binancecoin",
    "symbol": "bnb",
    "name": "Binance Coin",
    "image": "https://assets.coingecko.com/coins/images/825/large/binance-coin-logo
    "current_price": 212.03,
    "market_cap": 33015186690,
    "total_volume": 2490184836,
    "high_24h": 230.59,
    "low_24h": 210.87,
  },
  {
    "id": "bitcoin",
    "symbol": "btc",
    "name": "Bitcoin",
    "image": "https://assets.coingecko.com/coins/images/1/large/bitcoin.png?1547033
    "current_price": 47553,
    "market_cap": 901453728232,
    "total_volume": 47427138554,
    "high_24h": 51131,
    "low_24h": 48056,
  },
  {
    "id": "cardano",
    "symbol": "ada",
    "name": "Cardano",
    "image": "https://assets.coingecko.com/coins/images/975/large/cardano.png?15470
    "current_price": 0.84514,
    "market_cap": 27210647217,
    "total_volume": 3204270671,
    "high_24h": 0.919055,
    "low_24h": 0.843236,
  },
  {
    "id": "ethereum",
    "symbol": "eth",
    "name": "Ethereum",
    "image": "https://assets.coingecko.com/coins/images/279/large/ethereum.png?1595
    "current_price": 1479.97,
    "market_cap": 172447578072,
    "total_volume": 24709055087,
    "high_24h": 1597.13,
    "low_24h": 1493,
  },
  {
    "id": "litecoin",
    "symbol": "ltc",
    "name": "Litecoin",
    "image": "https://assets.coingecko.com/coins/images/2/large/litecoin.png?154703
    "current_price": 171.49,
    "market_cap": 11561005268,
    "total_volume": 4950077782,
    "high_24h": 187.34,
    "low_24h": 172.45,
  },
  {
    "id": "polkadot",
    "symbol": "dot",
```

```
    "name": "Polkadot",
    "image": "https://assets.coingecko.com/coins/images/12171/large/aJGBjJFU_400x40
    "current_price": 29.28,
    "market_cap": 28856989783,
    "total_volume": 1266769267,
    "high_24h": 32.2,
    "low_24h": 29.54,
  },
  {
    "id": "ripple",
    "symbol": "xrp",
    "name": "XRP",
    "image": "https://assets.coingecko.com/coins/images/44/large/xrp-symbol-white-1
    "current_price": 0.360658,
    "market_cap": 16580549437,
    "total_volume": 2357746464,
    "high_24h": 0.381072,
    "low_24h": 0.358941,
  },
  {
    "id": "tether",
    "symbol": "usdt",
    "name": "Tether",
    "image": "https://assets.coingecko.com/coins/images/325/large/Tether-logo.png?1
    "current_price": 0.83869,
    "market_cap": 32307660438,
    "total_volume": 82854947322,
    "high_24h": 0.843104,
    "low_24h": 0.832594,
  },
  {
    "id": "uniswap",
    "symbol": "uni",
    "name": "Uniswap",
    "image": "https://assets.coingecko.com/coins/images/12504/large/uniswap-uni.png
    "current_price": 24.94,
    "market_cap": 13099199643,
    "total_volume": 939432128,
    "high_24h": 27.92,
    "low_24h": 24.78,
  }
]
```


https://docs.python.org/3/library/functions.html#sorted (https://docs.python.org/3/library/functions.html#sorted)

```
sorted(iterable, *, key=None, reverse=False)
```

V dokumentaciji vidimo, da lahko kontroliramo katere vrednosti primerjamo z uporabo **key** parametra.

Kot **key** lahko podamo našo funkcijo, ki sprejme 1 argument in vrne vrednost po kateri primerjamo.

In [22]:

```
def sort_funkcija(x):
    print(f'{x["id"]}    \t {x["market_cap"]}')
    return x["market_cap"]

sorted(data, key=sort_funkcija, reverse=True)
```

```
binancecoin      33015186690
bitcoin          901453728232
cardano          27210647217
ethereum         172447578072
litecoin         11561005268
polkadot         28856989783
ripple           16580549437
tether           32307660438
uniswap          13099199643
```

Out[22]:


```
[{'id': 'bitcoin',
  'symbol': 'btc',
  'name': 'Bitcoin',
  'image': 'https://assets.coingecko.com/coins/images/1/large/bitcoi
n.png?1547033579',
  'current_price': 47553,
  'market_cap': 901453728232,
  'total_volume': 47427138554,
  'high_24h': 51131,
  'low_24h': 48056},
 {'id': 'ethereum',
  'symbol': 'eth',
  'name': 'Ethereum',
  'image': 'https://assets.coingecko.com/coins/images/279/large/ethe
reum.png?1595348880',
  'current_price': 1479.97,
  'market_cap': 172447578072,
  'total_volume': 24709055087,
  'high_24h': 1597.13,
  'low_24h': 1493},
 {'id': 'binancecoin',
  'symbol': 'bnb',
  'name': 'Binance Coin',
  'image': 'https://assets.coingecko.com/coins/images/825/large/bina
nce-coin-logo.png?1547034615',
  'current_price': 212.03,
  'market_cap': 33015186690,
  'total_volume': 2490184836,
  'high_24h': 230.59,
  'low_24h': 210.87},
 {'id': 'tether',
  'symbol': 'usdt',
  'name': 'Tether',
  'image': 'https://assets.coingecko.com/coins/images/325/large/Teth
er-logo.png?1598003707',
  'current_price': 0.83869,
  'market_cap': 32307660438,
  'total_volume': 82854947322,
  'high_24h': 0.843104,
  'low_24h': 0.832594},
 {'id': 'polkadot',
```

```
   'symbol': 'dot',
   'name': 'Polkadot',
   'image': 'https://assets.coingecko.com/coins/images/12171/large/aJ
GBjJFU_400x400.jpg?1597804776',
   'current_price': 29.28,
   'market_cap': 28856989783,
   'total_volume': 1266769267,
   'high_24h': 32.2,
   'low_24h': 29.54},
  {'id': 'cardano',
   'symbol': 'ada',
   'name': 'Cardano',
   'image': 'https://assets.coingecko.com/coins/images/975/large/card
ano.png?1547034860',
   'current_price': 0.84514,
   'market_cap': 27210647217,
   'total_volume': 3204270671,
   'high_24h': 0.919055,
   'low_24h': 0.843236},
  {'id': 'ripple',
   'symbol': 'xrp',
   'name': 'XRP',
   'image': 'https://assets.coingecko.com/coins/images/44/large/xrp-s
ymbol-white-128.png?1605778731',
   'current_price': 0.360658,
   'market_cap': 16580549437,
   'total_volume': 2357746464,
   'high_24h': 0.381072,
   'low_24h': 0.358941},
  {'id': 'uniswap',
   'symbol': 'uni',
   'name': 'Uniswap',
   'image': 'https://assets.coingecko.com/coins/images/12504/large/un
iswap-uni.png?1600306604',
   'current_price': 24.94,
   'market_cap': 13099199643,
   'total_volume': 939432128,
   'high_24h': 27.92,
   'low_24h': 24.78},
  {'id': 'litecoin',
   'symbol': 'ltc',
   'name': 'Litecoin',
   'image': 'https://assets.coingecko.com/coins/images/2/large/liteco
in.png?1547033580',
   'current_price': 171.49,
   'market_cap': 11561005268,
   'total_volume': 4950077782,
   'high_24h': 187.34,
   'low_24h': 172.45}]
```


Isto sortiranje lahko dobimo z uporabo lambda funkcije.

In [23]:

```
sorted(data, key=lambda x: x["market_cap"], reverse=True)
```

Out[23]:


```
[{'id': 'bitcoin',
  'symbol': 'btc',
  'name': 'Bitcoin',
  'image': 'https://assets.coingecko.com/coins/images/1/large/bitcoin.
png?1547033579',
  'current_price': 47553,
  'market_cap': 901453728232,
  'total_volume': 47427138554,
  'high_24h': 51131,
  'low_24h': 48056},
 {'id': 'ethereum',
  'symbol': 'eth',
  'name': 'Ethereum',
  'image': 'https://assets.coingecko.com/coins/images/279/large/ethere
um.png?1595348880',
  'current_price': 1479.97,
  'market_cap': 172447578072,
  'total_volume': 24709055087,
  'high_24h': 1597.13,
  'low_24h': 1493},
 {'id': 'binancecoin',
  'symbol': 'bnb',
  'name': 'Binance Coin',
  'image': 'https://assets.coingecko.com/coins/images/825/large/binanc
e-coin-logo.png?1547034615',
  'current_price': 212.03,
  'market_cap': 33015186690,
  'total_volume': 2490184836,
  'high_24h': 230.59,
  'low_24h': 210.87},
 {'id': 'tether',
  'symbol': 'usdt',
  'name': 'Tether',
  'image': 'https://assets.coingecko.com/coins/images/325/large/Tether
-logo.png?1598003707',
  'current_price': 0.83869,
  'market_cap': 32307660438,
  'total_volume': 82854947322,
  'high_24h': 0.843104,
  'low_24h': 0.832594},
 {'id': 'polkadot',
  'symbol': 'dot',
  'name': 'Polkadot',
  'image': 'https://assets.coingecko.com/coins/images/12171/large/aJGB
jJFU_400x400.jpg?1597804776',
  'current_price': 29.28,
  'market_cap': 28856989783,
  'total_volume': 1266769267,
  'high_24h': 32.2,
  'low_24h': 29.54},
 {'id': 'cardano',
  'symbol': 'ada',
  'name': 'Cardano',
  'image': 'https://assets.coingecko.com/coins/images/975/large/cardan
o.png?1547034860',
```

```
   'current_price': 0.84514,
   'market_cap': 27210647217,
   'total_volume': 3204270671,
   'high_24h': 0.919055,
   'low_24h': 0.843236},
  {'id': 'ripple',
   'symbol': 'xrp',
   'name': 'XRP',
   'image': 'https://assets.coingecko.com/coins/images/44/large/xrp-sym
bol-white-128.png?1605778731',
   'current_price': 0.360658,
   'market_cap': 16580549437,
   'total_volume': 2357746464,
   'high_24h': 0.381072,
   'low_24h': 0.358941},
  {'id': 'uniswap',
   'symbol': 'uni',
   'name': 'Uniswap',
   'image': 'https://assets.coingecko.com/coins/images/12504/large/unis
wap-uni.png?1600306604',
   'current_price': 24.94,
   'market_cap': 13099199643,
   'total_volume': 939432128,
   'high_24h': 27.92,
   'low_24h': 24.78},
  {'id': 'litecoin',
   'symbol': 'ltc',
   'name': 'Litecoin',
   'image': 'https://assets.coingecko.com/coins/images/2/large/litecoi
n.png?1547033580',
   'current_price': 171.49,
   'market_cap': 11561005268,
   'total_volume': 4950077782,
   'high_24h': 187.34,
   'low_24h': 172.45}]
```

Naloga:

Imamo podatke o GDP Evropskih držav od leta 2010 do 2020.

Uporabite funkcijo **sorted()** in določite takšno **lambda funkcijo**, da razvrstimo države po GDP leta 2020 od največje do najmanjše.

Izpišite imena držav od največje do najmanjše.

Primeri:

```
Input:
data = [["Austria", 392.623, 431.515, 409.652, 430.203, 442.698, 381.998, 3
94.215, 417.721, 456.166, 447.718, 432.894],
["Belgium", 484.450, 527.492,498.161, 521.090, 531.651, 456.067, 469.931, 4
95.953, 532.268, 517.609, 503.416],
["Bosnia", 17.164, 18.629, 17.207, 18.155, 18.522, 16.210, 16.910, 18.081,
20.162, 20.106, 18.893],
["Bulgaria", 50.611, 57.420, 53.901, 55.557, 56.815, 50.201, 53.236, 58.342
, 65.197, 66.250, 67.917],
["Croatia", 59.866, 62.399, 56.549, 58.158, 57.683, 49.519, 51.623, 55.201,
60.805, 60.702, 56.768],
["Cyprus", 25.608, 27.454, 25.055, 24.094, 23.401, 19.691, 20.461, 22.189,
24.493, 24.280, 23.246],
["Czech Republic", 207.478, 227.948, 207.376, 209.402, 207.818, 186.830, 19
5.090, 215.914, 245.226, 246.953, 241.975],
["Denmark",321.995,344.003,327.149,343.584, 352.994,302.673,311.988,329.866
,352.058,347.176,339.626],
["Estonia",19.536,23.191,23.057,25.145, 26.658, 22.916,23.994,26.850,30.761
,31.038,30.468],
["Finland",248.262,273.925,256.849,270.065,273.042,232.582,239.150,252.867,
274.210,269.654,267.856],
["France",2647.537,2864.030,2685.311,2811.957,2856.697,2439.435,2466.152,25
91.775,2780.152,2707.074,2551.451],
["Germany",3423.466,3761.142,3545.946,3753.687,3904.921,3383.091,3496.606,3
664.511,3951.340,3863.344,3780.553],
["Greece",299.919,288.062,245.807,239.937,237.406,196.690,195.303,203.493,2
18.230,214.012,194.376],
["Hungary",130.923,140.782,127.857,135.221,140.083,123.074,126.008, 139.844
,161.182,170.407,149.939],
["Iceland",13.684,15.159,14.724,16.034, 17.758,17.389,20.618,24.457,25.965,
23.918,20.805],
["Ireland",222.533,238.088, 225.140,238.708,259.200,290.858,301.968,335.211
,382.754,384.940,399.064],
["Italy",2129.021,2278.376,2073.971,2131.159,2155.151,1833.195,1869.973,195
0.703,2075.856,2001.440,1848.222],
["Latvia",23.809,28.496,28.141,30.260,31.385,26.986,27.707,30.528,34.882,3
5.045,33.015],
["Liechtenstein",5.082,5.740,5.456,6.392,6.657,6.268,6.215],
["Lithuania",37.200,43.564, 42.887,46.423,48.632,41.538,42.991,47.645,53.30
2,53.641,55.064],
["Luxembourg",53.312,60.060,56.709,61.759,66.209,57.233,58.985,62.449,69.55
3,69.453,68.613],
["Malta",8.757,9.511,9.215,10.154,11.302,10.701,11.446,12.764,14.560,14.859
,14.290],
["Montenegro",4.147,4.543,4.090,4.466,4.595,4.055,4.376,4.855,5.457,5.424,
4.943],
["Netherlands",848.133,904.915,839.436, 877.198,892.397,765.650,783.852,83
3.575,914.519,902.355,886.339],
["Norway",429.131,498.832,510.229,523.502,499.338,386.663,371.345,398.394,4
34.167,417.627,366.386],
["Poland",479.161,528.571,500.846,524.399,545.284,477.568,471.843,526.749,5
85.816,565.854,580.894],
```

```
["Portugal",238.748,245.119,216.488,226.144,229.995,199.521,206.361,221.280
,240.901,236.408,221.716],
["Romania",166.225, 183.443,171.196,190.948,199.628,177.895,188.495,211.407
,239.552,243.698,248.624],
["Serbia",41.369,49.280,43.300, 48.394,47.062,39.629,40.630,44.120, 50.509,
51.523,51.999],
["Slovakia",89.668, 98.271,93.466,98.509,101.109,87.814,89.885,95.821,106.5
73,106.552,101.892],
["Slovenia",48.103,51.338,46.378,48.131,49.969,43.124,44.660,48.545,54.059,
54.154,51.802],
["Spain",1434.286,1489.431,1336.759,1362.280,1379.098,1199.688,1238.010, 13
17.104,1427.533,1397.870,1247.464],
["Sweden",488.909,563.797,544.482,579.361,574.413,498.118,512.205,540.545,5
56.073,528.929,529.054],
["Switzerland",583.053, 699.670,667.890,688.747,709.496,679.721,670.247,68
0.029,705.546,715.360,707.868],
["Turkey",772.290,832.497,873.696,950.328,934.075,859.449,863.390,852.648,7
71.274,743.708,649.436],
["United Kingdom",2455.309,2635.799,2677.082,2755.356,3036.310,2897.060,266
9.107,2640.067,2828.833,2743.586,2638.296]]
```

```
Output:
Germany
United Kingdom
France
Italy
Spain
Netherlands
Switzerland
Turkey
Poland
Sweden
Belgium
Austria
Ireland
Norway
Denmark
Finland
Romania
Czech Republic
Portugal
Greece
Hungary
Slovakia
Luxembourg
Bulgaria
Croatia
Lithuania
Serbia
Slovenia
```

```
Latvia
Estonia
Cyprus
Iceland
Bosnia
Malta
Liechtenstein
Montenegro
```

In [96]:

```
data = [["Austria", 392.623, 431.515, 409.652, 430.203, 442.698, 381.998, 394.215,
["Belgium", 484.450, 527.492,498.161, 521.090, 531.651, 456.067, 469.931, 495.953,
["Bosnia", 17.164, 18.629, 17.207, 18.155, 18.522, 16.210, 16.910, 18.081, 20.162,
["Bulgaria", 50.611, 57.420, 53.901, 55.557, 56.815, 50.201, 53.236, 58.342, 65.197
["Croatia", 59.866, 62.399, 56.549, 58.158, 57.683, 49.519, 51.623, 55.201, 60.805,
["Cyprus", 25.608, 27.454, 25.055, 24.094, 23.401, 19.691, 20.461, 22.189, 24.493,
["Czech Republic", 207.478, 227.948, 207.376, 209.402, 207.818, 186.830, 195.090, 2
["Denmark",321.995,344.003,327.149,343.584, 352.994,302.673,311.988,329.866,352.058
["Estonia",19.536,23.191,23.057,25.145, 26.658, 22.916,23.994,26.850,30.761,31.038,
["Finland",248.262,273.925,256.849,270.065,273.042,232.582,239.150,252.867,274.210,
["France",2647.537,2864.030,2685.311,2811.957,2856.697,2439.435,2466.152,2591.775,2
["Germany",3423.466,3761.142,3545.946,3753.687,3904.921,3383.091,3496.606,3664.511,
["Greece",299.919,288.062,245.807,239.937,237.406,196.690,195.303,203.493,218.230,2
["Hungary",130.923,140.782,127.857,135.221,140.083,123.074,126.008, 139.844,161.182
["Iceland",13.684,15.159,14.724,16.034, 17.758,17.389,20.618,24.457,25.965, 23.918,
["Ireland",222.533,238.088, 225.140,238.708,259.200,290.858,301.968,335.211,382.754
["Italy",2129.021,2278.376,2073.971,2131.159,2155.151,1833.195,1869.973,1950.703,20
["Latvia",23.809,28.496,28.141,30.260,31.385,26.986,27.707,30.528,34.882,35.045,33.
["Lithuania",37.200,43.564, 42.887,46.423,48.632,41.538,42.991,47.645,53.302,53.641
["Luxembourg",53.312,60.060,56.709,61.759,66.209,57.233,58.985,62.449,69.553,69.453
["Malta",8.757,9.511,9.215,10.154,11.302,10.701,11.446,12.764,14.560,14.859,14.290]
["Montenegro",4.147,4.543,4.090,4.466,4.595,4.055,4.376,4.855,5.457,5.424,4.943],
["Netherlands",848.133,904.915,839.436, 877.198,892.397,765.650,783.852,833.575,914
["Norway",429.131,498.832,510.229,523.502,499.338,386.663,371.345,398.394,434.167,4
["Poland",479.161,528.571,500.846,524.399,545.284,477.568,471.843,526.749,585.816,5
["Portugal",238.748,245.119,216.488,226.144,229.995,199.521,206.361,221.280,240.901
["Romania",166.225, 183.443,171.196,190.948,199.628,177.895,188.495,211.407,239.552
["Serbia",41.369,49.280,43.300, 48.394,47.062,39.629,40.630,44.120, 50.509,51.523,5
["Slovakia",89.668, 98.271,93.466,98.509,101.109,87.814,89.885,95.821,106.573,106.5
["Slovenia",48.103,51.338,46.378,48.131,49.969,43.124,44.660,48.545,54.059,54.154,5
["Spain",1434.286,1489.431,1336.759,1362.280,1379.098,1199.688,1238.010, 1317.104,1
["Sweden",488.909,563.797,544.482,579.361,574.413,498.118,512.205,540.545,556.073,5
["Switzerland",583.053, 699.670,667.890,688.747,709.496,679.721,670.247,680.029,705
["Turkey",772.290,832.497,873.696,950.328,934.075,859.449,863.390,852.648,771.274,7
["United Kingdom",2455.309,2635.799,2677.082,2755.356,3036.310,2897.060,2669.107,26
```

In [97]:

```
[e[0] for e in sorted(data,key=lambda x : x[-1],reverse = True)]
```

Out[97]:

```
['Germany',
 'United Kingdom',
 'France',
 'Italy',
 'Spain',
 'Netherlands',
 'Switzerland',
 'Turkey',
 'Poland',
 'Sweden',
 'Belgium',
 'Austria',
 'Ireland',
 'Norway',
 'Denmark',
 'Finland',
 'Romania',
 'Czech Republic',
 'Portugal',
 'Greece',
 'Hungary',
 'Slovakia',
 'Luxembourg',
 'Bulgaria',
 'Croatia',
 'Lithuania',
 'Serbia',
 'Slovenia',
 'Latvia',
 'Estonia',
 'Cyprus',
 'Iceland',
 'Bosnia',
 'Malta',
 'Montenegro']
```

In [40]:

```
data_sorted = sorted(data, key=lambda x: x[-1], reverse=True)
for i in data_sorted:
    print(i[0])
```

```
Germany
United Kingdom
France
Italy
Spain
Netherlands
Switzerland
Turkey
Poland
Sweden
Belgium
Austria
Ireland
Norway
Denmark
Finland
Romania
Czech Republic
Portugal
Greece
Hungary
Slovakia
Luxembourg
Bulgaria
Croatia
Lithuania
Serbia
Slovenia
Latvia
Estonia
Cyprus
Iceland
Bosnia
Malta
Montenegro
```

In [ ]:

```
```

Variable scope

Spremenljivke se razlikujejo tudi po tem koliko dolgo obstajajo (variable lifetime) in od kje lahko dostopamo do njih (variable scope).

Spremenljivka definirana znotraj funkcije (kot parameter ali navadno) obstaja samo znotraj funkcije.

Ko se izvajanje funkcije konča, spremenljivka neha obstajati.

In [72]:

```
def funkcija(spr1):
    spr2 = 10
    print(f"Spr1: {spr1}")
    print(f"Spr2: {spr2}")


funkcija(5)
print(f"Spr1: {spr1}")
print(f"Spr2: {spr2}")
```

```
Spr1: 5
Spr2: 10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
 last)
<ipython-input-72-d9649ca9516e> in <module>
      6
      7 funkcija(5)
----> 8 print(f"Spr1: {spr1}")
      9 print(f"Spr2: {spr2}")

NameError: name 'spr1' is not defined
```

Spremenljivka definirana znotraj naše glavne kode (zunaj naših funkcij) je **globalna spremenljivka** in je dostopna skozi našo celotno kodo.

In [73]:

```
spr1 = 5
print(f"Spr1: {spr1}")

if spr1 == 5:
    spr2 = 10
print(f"Spremenljivka2: {spr2}")
print()

def funkcija():
    spr3 = 200
    print(f"Spr1: {spr1}")
    print(f"Spr2: {spr2}")
    print(f"Spr3: {spr3}")

funkcija()
print()

print(f"Spr1: {spr1}")
print(f"Spr2: {spr2}")
```

```
Spr1: 5
Spremenljivka2: 10

Spr1: 5
Spr2: 10
Spr3: 200

Spr1: 5
Spr2: 10
```

Problem se lahko pojavi, če znotraj funkcije definiramo spremenljivko z enakim imenom, ki že obstaja kot globalna spremenljivka.

V tem primeru bo python spremenljivki označil kot dve različni spremenljivki. Ena dostopna znotraj funkcije, druga dostopna zunaj funkcije.

In [164]:

```
spr1 = 5
print(f"Spr1: {spr1}")

def funkcija():
    spr1 = 100
    print(f"Spr1: {spr1}")

funkcija()
print(f"Spr1: {spr1}")
```

```
Spr1: 5
Spr1: 100
Spr1: 5
```

Parameter se obnaša kot lokalna spremenljivka.

In [175]:

```
spr1 = 5
print(f"Spr1: {spr1}")

def funkcija(spr1):
    print(f"Spr1: {spr1}")

funkcija(100)
print(f"Spr1: {spr1}")
```

```
Spr1: 5
Spr1: 100
Spr1: 5
```

Paziti je potrebno, ko posredujemo list ali dictionary kot argument.

In [74]:

```
def funkcija(l):
    print(l)
    l[0] = 100

seznam = [3, 7, 13]
funkcija(seznam)
print(seznam)
```

```
[3, 7, 13]
[100, 7, 13]
```

In [75]:

```
def funkcija(d):
    print(d)
    d["a"] = 100

dict_ = {"a": 5, "b": 6, "c": 7}
funkcija(dict_)
print(dict_)
```

```
{'a': 5, 'b': 6, 'c': 7}
{'a': 100, 'b': 6, 'c': 7}
```

In [ ]:

```

```

Če želimo spreminjati globalno spremenljivko znotraj funkcije (znotraj local scope) moramo uporabiti besedo **global**.

In [76]:

```
spr1 = 5
print(f"Spr1: {spr1}")

def funkcija():
    global spr1
    spr1 = 100
    print(f"Spr1: {spr1}")

funkcija()
print(f"Spr1: {spr1}")
```

```
Spr1: 5
Spr1: 100
Spr1: 100
```

S to besedo lahko tudi ustvarimo novo globalno spremenljivko, znotraj localnega scopa.

In [77]:

```
def funkcija():
    global spr1
    spr1 = 5
    print(f"Spr1: {spr1}")

funkcija()
print(f"Spr1: {spr1}")
```

```
Spr1: 5
Spr1: 5
```

In [ ]:

```

```

Naloga:

Napišite funkcijo, kjer lahko igramo **vislice**.

Funkcija **vislice()** naj ima 2 parametra. Prvi je besedo katero se ugiba in drugi število možnih ugibov. Če števila ugibov ne podamo naj bo default vrednost 10.

Uporabnika konstantno sprašujte naj vnese črko. Nato izpišite iskano besedo. Črke katere je uporabnik uganil izpišite normalno, črke katere še ni uganil pa nadomestite z _.

Dodatno zraven prikazujte katere vse črke je uporabnik že preizkusil.

Če uporabnik besedo uspešno ugani v danih poizkusih naj funkcija vrne vrednost True. V nasprotnem primeru naj vrne vrednost False.

Primeri:

```
Input:
vislice("jabolko")

Output:
Guesses so far [].
What is your guess? a
_ a_ _ _ _ _

Guesses so far ['a'].
What is your guess? e
_ a_ _ _ _ _

Guesses so far ['a', 'e'].
What is your guess? o
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o'].
What is your guess? p
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p'].
What is your guess? r
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p', 'r'].
What is your guess? l
_ a_ ol_ o

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l'].
What is your guess? k
_ a_ olko

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k'].
What is your guess? j
ja_ olko

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k', 'j'].
What is your guess? b
jabolko
KONEC
True
```

In [207]:

```
# Rešitev
def vislice(beseda, n=10):
    correct_guesses = []
    all_guesses = []

    try_ = 0
    while try_ < n:
        print()
        guess = input(f"Guesses so far {all_guesses}. \nWhat is your guess? ")
        all_guesses.append(guess)
        if guess in beseda:
            correct_guesses.append(guess)

        beseda_print = ""
        for ch in beseda:
            if ch in correct_guesses:
                beseda_print += ch
            else:
                beseda_print += "_ "
        print(beseda_print)
        if len(set(correct_guesses)) == len(set(beseda)):
            print("KONEC")
            return True

        try_ += 1

    return False


print(vislice("jabolko"))
```

```
Guesses so far [].
What is your guess? a
_ a_ _ _ _ _

Guesses so far ['a'].
What is your guess? e
_ a_ _ _ _ _

Guesses so far ['a', 'e'].
What is your guess? o
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o'].
What is your guess? p
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p'].
What is your guess? r
_ a_ o_ _ o

Guesses so far ['a', 'e', 'o', 'p', 'r'].
What is your guess? l
_ a_ ol_ o

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l'].
What is your guess? k
_ a_ olko
```

```
Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k'].
What is your guess? j
ja_ olko

Guesses so far ['a', 'e', 'o', 'p', 'r', 'l', 'k', 'j'].
What is your guess? b
jabolko
KONEC
True
```

In [ ]:

```
```

Naloga:

Ustvarite program **Križci in Krožci**

Igralno polje lahko predstavite kot liste znotraj lista, kjer *E* predstavlja prazno polje.

```
board = [["X", "E", "E"],
         ["O", "E", "E"],
         ["E", "E", "E"]]
```

Od igralcev nato izmenično zahtevajte polje v katerega želijo postaviti svoj znak. Privzememo lahko, da bodo igralci igrali pravično in vpisovali samo prazna polja.

Primeri:

```
Output:
['E', 'E', 'E']
['E', 'E', 'E']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '00

['X', 'E', 'E']
['E', 'E', 'E']
['E', 'E', 'E']
It's O's turn. Make a move (exp: 12): '12

['X', 'E', 'E']
['E', 'E', 'O']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '10

['X', 'E', 'E']
['X', 'E', 'O']
['E', 'E', 'E']
It's O's turn. Make a move (exp: 12): '12

['X', 'E', 'E']
['X', 'E', 'O']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '20
X je ZMAGOVALEC!
```

9:15 - 9:25 make move

In [92]:

```
def display_board(board):
    for row in board:
        print(row)

def make_move(on_turn, board):
    move = input(f"It's {on_turn}'s turn. Make a move (exp: 12): '")
    row = int(move[0])
    col = int(move[1])
    board[row][col] = on_turn

def is_game_over(board):
    # pregled po vrsticah
    for row in board:
        if row[0] != "E":
            if row[0] == row[1] and row[0] == row[2]:
                return True
    # pregled po stolpcih
    for i in range(3):
        if board[0][i] != "E":
            if board[0][i] == board[1][i] and board[0][i] == board[2][i]:
                return True
    # pregled ene diagonale
    if board[0][0] != "E":
        if board[0][0] == board[1][1] and board[0][0] == board[2][2]:
            return True
    # pregled druge diagonale
    if board[0][2] != "E":
        if board[0][2] == board[1][1] and board[0][2] == board[2][0]:
            return True

    return False

def play():
    board = [["E", "E", "E"],
             ["E", "E", "E"],
             ["E", "E", "E"]]
    on_turn = "X"
    while True:
        display_board(board)
        make_move(on_turn, board)

        game_over = is_game_over(board)
        if game_over:
            print(f"{on_turn} je ZMAGOVALEC!")
            break
        else:
            if on_turn == "X":
                on_turn = "O"
            elif on_turn == "O":
                on_turn = "X"
        print()


play()
```

```
['E', 'E', 'E']
['E', 'E', 'E']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '00
```

```
Changing players

['X', 'E', 'E']
['E', 'E', 'E']
['E', 'E', 'E']
It's O's turn. Make a move (exp: 12): '12
Changing players

['X', 'E', 'E']
['E', 'E', 'O']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '10
Changing players

['X', 'E', 'E']
['X', 'E', 'O']
['E', 'E', 'E']
It's O's turn. Make a move (exp: 12): '12
Changing players

['X', 'E', 'E']
['X', 'E', 'O']
['E', 'E', 'E']
It's X's turn. Make a move (exp: 12): '20
X je ZMAGOVALEC!
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```