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
```

Naloga:

Napiši funkcijo, ki sprejme 3 argumente (integer števila).

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
```

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]:

```python
# 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_grafa}.")

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'>
```

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]
```

In [ ]:

```

```

In [ ]:

```

```

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
```

Naloga:

Ustvarite funkcijo, ki kot parametra vzeme list števil 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 [1]:

```
def fun_03(list_, ukaz="asc"):
    new_list = []

    if ukaz == "asc":
        while list_:
            minimum = min(list_)
            new_list.append(minimum)
            list_.remove(minimum)
    elif ukaz == "desc":
        print("Razvrsti od največje do najumanjše")
        while list_:
            maximum = max(list_)
            new_list.append(maximum)
            list_.remove(maximum)
    else:
        print("Napačna vrednost!")
    return new_list

print(fun_03([1,4,2,8,4,0], ukaz="desc"))
#[8, 4, 4, 2, 1, 0]

print(fun_03([1,4,2,8,4,0], ukaz="asc"))
#[0, 1, 2, 4, 4, 8]

print(fun_03([5,8,-2,13,6,-6]))
# [-6, -2, 5, 6, 8, 13]
```

```
Razvrsti od največje do najumanjše
[8, 4, 4, 2, 1, 0]
[0, 1, 2, 4, 4, 8]
[-6, -2, 5, 6, 8, 13]
```

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/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}]
```

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 [ ]:

```

```

Generators

Generatorji so funkcije namenjene generiranju iteratorjev (objekti, ki so lahko iterirani - list, itd..).

Razlika je, da generatorji generiranjo vrednosti eno po eno, ne vse naenkrat, kar jih nrdi veliko bolj memory-efficient.

Ustvarimo jih enako kot navadno funkcijo, le da namesto `return` uporbimo `yield` .

`yield` pavzira funkcijo in shrani njeno stanje, tako da lahko kasneje nadaljujemo kjer smo končali.

In [57]:

```
def moj_range(n):
    print("Start creating moj range")
    while n<10:
        yield n
        n += 1
    print("Stop generator")

val = moj_range(5)
print(val)
print(type(val))
```

```
<generator object moj_range at 0x000001D59110D740>
<class 'generator'>
```

Ko prvič pokličemo `next()` se program začne izvajati na začetku funkcije in nadaljuje do `yield` kjer vrne vrednost.

Naslednji klici `next()` nadaljujejo izvajanje programa od `yield` naprej do naslednjega `yield` .

Če ne naleti na `yield` dvigne StopIteration exception.

In [58]:

```
print(next(val))
```

```
Start creating moj range
5
```

In [59]:

```
next(val)
```

Out[59]:

```
6
```

In [60]:

```
next(val)
```

Out[60]:

```
7
```

In [61]:

```
#val = moj_range(5) # Če vmes ponovno kličemo generator bo šlo od začetka.
```

In [62]:

```
next(val)
```

Out[62]:

```
8
```

In [63]:

```
next(val)
```

Out[63]:

```
9
```

In [64]:

```
next(val)
```

```
Stop generator

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call
 last)
<ipython-input-64-a2a2bf9708c5> in <module>
----> 1 next(val)

StopIteration:
```

Z generatorjem lahko ustvarimo svojo `range()` funkcionalnost.

In [67]:

```
def moj_range(n, m, step=1):
    while n<m:
        yield n
        n+=step

print("Primer: moj_range")
for i in moj_range(1, 20, 2):
    print(i)


print("Primer: range()")
for i in range(1, 20, 2):
    print(i)
```

```
Primer: moj_range
1
3
5
7
9
11
13
15
17
19
Primer: range()
1
3
5
7
9
11
13
15
17
19
```

In [ ]:

```
```

Comprehensions

Poleg generatorjev, lahko za kreiranje listov uporabimo tudi **list comprehensions**.

- List comprehensions so bolj berljivi od built-in funkcij, ki potrebujejo lambda expressions
- List comprehensions nam dovolijo filtriranje elementov

In [69]:

```
# Primer: želimo narediti list kvadratov iz lista a
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [x**2 for x in a]
print(a)
print(squares)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [70]:

```
# Primer: Filtriranje elementov
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

even_squares = [x**2 for x in a if x%2 == 0 and x%3==0]
print(a)
print(even_squares)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
[36, 144]
```

set comprehensions

In [85]:

```
a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
even_squares = {x**2 for x in a if x % 2 == 0}
print(even_squares)
print(type(even_squares))
```

```
{64, 100, 4, 36, 16}
<class 'set'>
```

Dictionary Comprehensions

```
ict_variable = {key:value for (key,value) in dictonary.items()}
```

In [86]:

```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
# Double each value in the dictionary
double_dict1 = {k:v*2 for (k,v) in dict1.items()}
print(double_dict1)
```

```
{'a': 2, 'b': 4, 'c': 6, 'd': 8, 'e': 10}
```

Generator Expressions

Podobno kot list comprehensions lahko zapišemo tudi generatorje. Razlika je, da oni vrnejo generator objekt in ne list-e.

In [71]:

```
import sys

my_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

list_comprehension = [x for x in my_list]
set_comprehension = {x for x in my_list}
gen_expression = (x for x in my_list) # the language name for these is generator ex

print(list_comprehension)
print(type(list_comprehension))
print(sys.getsizeof(list_comprehension))
print()

print(set_comprehension)
print(type(set_comprehension))
print(sys.getsizeof(set_comprehension))
print()

print(gen_expression)
print(type(gen_expression))
print(sys.getsizeof(gen_expression))
for val in gen_expression:
    print(val)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
<class 'list'>
184

{'d', 'j', 'f', 'g', 'i', 'a', 'b', 'c', 'e', 'h'}
<class 'set'>
728

<generator object <genexpr> at 0x000001D5910FE900>
<class 'generator'>
112
a
b
c
d
e
f
g
h
i
j
```

Glavna razlika med `generator expressions` in `list comprehension` je, da so generatorji počasnejši ampak prišparajo na spominu.

In [ ]: