P3_Prenovljeno_Funkcije-Copy1

May 24, 2023

1 Križci in Krožci

Naslednji program katerega bomo napisali je igra **Križci in Krožci**.

Primer igranja igre:

```
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!
```

Uprašat, **kakšni bi bili koraki, da naredimo Križce in Krožce?**

- Rabimo prikazat ploščo
- Rabmo od uporabnika dobiti input kam hoče dat znak kaj še...?
- Rabmo preveriti ali je kdo zmagal, kaj še...?

Naš program bi okvirno izgledal sledeče:

```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče

# 1. Prikažemo igralno ploščo

# 2. Od igralca zahtevamo naj naredi svojo potezo

# 3. Preverimo ali je kdo zmagal

# 3.a Če je kdo zmagal, izpišemo zmagovalca

# 3.b Če ni zmagovalca gremo na korak 1
```

Ustvarimo **x__and__o.py** datoteko, v katero bomo pisali naš program.

Za začetek si poglejmo prikaz igralne plošče.

Ploščo lahko shranimo kot 2D list. Če je element `"X"` ali `"O"` potem je tja igralec dal svojo potezo. Če pa je prazno mesto ga pa prikažimo kot `"E"` (empty).

[1]:
```
# Prikaži igralno ploščo
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]
print(board)
```

```
[['E', 'E', 'E'], ['E', 'E', 'E'], ['E', 'E', 'E']]
```

S preprostim print ne dobimo lepega izpisa, zato uporabimo for loop.

[17]:
```
# Prikaži igralno ploščo
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

print("R\\C 0  1  2")
for i, row in enumerate(board):
    print(f"{i}   {row[0]}  {row[1]}  {row[2]}")
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
```

2 Križci Krožci

Naloga:

Dodajte še kodo, ki od uporabnika zahteva naj vnesejo v katero polje želijo postaviti svoj znak.

Vizualizacija kode

[12]:
```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

# 1. Prikažemo igralno ploščo
print("R\\C 0  1  2")
for i, row in enumerate(board):
    print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

# 2. Od igralca zahtevamo naj naredi svojo potezo
move = input(f"Make a move (R/C) (exp: 12): ")
row = int(move[0])
col = int(move[1])
board[row][col] = "X"

# 3. Preverimo ali je kdo zmagal

# 3.a Če je kdo zmagal, izpišemo zmagovalca

# 3.b Če ni zmagovalca gremo na korak 1

# Da lažje debuggiramo bomo dodal še tale prikaz plošče spet
print("R\\C 0  1  2")
for i, row in enumerate(board):
    print(f"{i}   {row[0]}  {row[1]}  {row[2]}")
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  X
2   E  E  E
```

Vidimo, da isti del kode večkrat uporabimo.

Če bi želeli spremeniti naš prikaz igralne plošče, bi to spremembo morali narediti na večih mestih. To lahko privede do napak v naši kodo. Zadevo lahko rešimo z zanko ampak pri bolj zapletenih programih koda hitra postane nepregledna.

Kar bi želeli je, da zapakiramo del kode, ki izpiše igralno plošno, v neko spremenljivko. Vsakič, ko kličemo spremenljivko bi sedaj izvedli ta del kode. Če bi želeli spremeniti, kako se izpiše igralna plošča, bi to lahko preprosto enkrat popravli.

Za to imamo v pythonu **funkcije**.

2.1 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 oglati 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 #   |
    ... #              |
    #
print("Nadaljevanje program")
```

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
```

[3]:
```
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.

[4]:
```
print("Začetek programa")
hello2()
print("Nadaljevanje programa")

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

```
Začetek programa
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [4], line 2
      1 print("Začetek programa")
----> 2 hello2()
      3 print("Nadaljevanje programa")
      5 def hello2():

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

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
```

[5]:
```
# Rešitev
def fun():
    emšo = input("Vnesi emšo: ")

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

```
fun()
```

```
Vnesi emšo: 213452343232
Star si 498 let
```

2.2 The pass statements

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

[6]:
```
if True:

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

```
  Cell In [6], line 3
    print("Hello") # should give IndentationError
    ^
IndentationError: expected an indented block after 'if' statement on line 1
```

[7]:
```
if True:
    pass
print("Hello") # should be fine now with the pass added
```

```
Hello
```

---

3 Križci Krožci

Naloga:

Implementirajte prikaz igralske plošče znotraj funkcije `display_board()`.

Spremenljivko `board` pustite zunaj funkcije.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  X
2   E  E  E
```

Vizualizacija kode

[13]:
```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

# 1. Prikažemo igralno ploščo
display_board()

# 2. Od igralca zahtevamo naj naredi svojo potezo
move = input(f"Make a move (R/C) (exp: 12): ")
row = int(move[0])
col = int(move[1])
board[row][col] = "X"

# 3. Preverimo ali je kdo zmagal

# 3.a Če je kdo zmagal, izpišemo zmagovalca

# 3.b Če ni zmagovalca gremo na korak 1

# Da lažje debuggiramo bomo dodal še tale prikaz plošče spet
display_board()
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  X
2   E  E  E
```

---

4 Križci Krožci

Naloga:

Implementirajte še potezo katero naredi igralec znotraj funkcije `make_move()`.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  X
2   E  E  E
```

Vizualizacija kode

[14]:
```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

def make_move():
    move = input(f"Make a move (R/C) (exp: 12): ")
    row = int(move[0])
    col = int(move[1])
    board[row][col] = "X"


# 1. Prikažemo igralno ploščo
display_board()

# 2. Od igralca zahtevamo naj naredi svojo potezo
make_move()

# 3. Preverimo ali je kdo zmagal

# 3.a Če je kdo zmagal, izpišemo zmagovalca

# 3.b Če ni zmagovalca gremo na korak 1

# Da lažje debuggiramo bomo dodal še tale prikaz plošče spet
display_board()
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
```

```
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  X
2   E  E  E
```

Problem je sedaj, ker vsakič vstavimo samo `X`. Kar bi želeli je, da v našo funkcijo `make_move()` pošljemo še znak od igralca kateri je na vrsti.

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 `( )`. “‘python def funkcija_1(x, y, z): # x, y, z are parameters pass

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

[15]:
```python
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!

[16]:
```python
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.

[17]:
```python
# 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)
Cell In [17], line 7
      4     print(f"Y vrednost: {y}")
      5     print(f"Z vrednost: {z}")
----> 7 funkcija_1(1, 2)

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

[18]:
```python
# 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)
Cell In [18], line 7
      4     print(f"Y vrednost: {y}")
      5     print(f"Z vrednost: {z}")
----> 7 funkcija_1(1, 2, 3, 4)

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

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
```

[19]:
```python
# 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.

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

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

[20]:
```python
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.

[21]:
```
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)
Cell In [21], line 4
      1 def pozdrav(naslavljanje, ime, priimek):
      2     print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")
----> 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.

[22]:
```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

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

```
Pozdravljeni gospod Miha Novak.
```

[23]:
```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

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

```
  Cell In [23], 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
```

[24]:
```
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.

[25]:
```
def funkcija(x, y, z=0):
    print(x + y + z)

funkcija(1, 2)
```

```
3
```

[26]:
```
def funkcija(x, y=0, z):
    print(x + y + z)

funkcija(1, 2, 3)
```

```
  Cell In [26], 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
```

[27]:
```
# 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
```

---

5 Križci Krožci

Naloga:

`make_move()` funkciji dodajte parameter `znak`, ki drži znak igralca.

Zaenkrat, ko kličemo funkcijo, `hard-code`ajmo znak.

Primeri:

```
display_board()
make_move("O")
display_board()
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  O
2   E  E  E
```

Vizualizacija kode

[30]:
```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

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


# 1. Prikažemo igralno ploščo
display_board()

# 2. Od igralca zahtevamo naj naredi svojo potezo
make_move("O")

# 3. Preverimo ali je kdo zmagal

# 3.a Če je kdo zmagal, izpišemo zmagovalca

# 3.b Če ni zmagovalca gremo na korak 1

# Da lažje debuggiramo bomo dodal še tale prikaz plošče spet
```

```
display_board()
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  O
2   E  E  E
```

---

6 Križci Krožci

Naloga:

Za konec dodajmo to vse v `while` zanko, ki se izvaja dokler obstaja še kakšna možna poteza.

Možna poteza obstaja, če obstaja `"E"` znotraj naše igralne plošče.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   E  E  E
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
```

```
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  O  E
It's O's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   O  O  O
1   E  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  O  O
1   O  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 20
```

Vizualizacija kode

[32]:
```python
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

def make_move(znak):
    print(f"It's {znak}'s turn.")
    move = input(f"Make a move (R/C) (exp: 12): ")
    row = int(move[0])
    col = int(move[1])
```

```
    board[row][col] = znak


while True:
    # 1. Prikažemo igralno ploščo
    display_board()

    # 2. Od igralca zahtevamo naj naredi svojo potezo
    make_move("O")

    # 3. Preverimo ali je kdo zmagal

    # 3.a Če je kdo zmagal, izpišemo zmagovalca

    # 3.b Če ni zmagovalca gremo na korak 1

    board_1D = []
    for row in board:
        board_1D.extend(row)
    if not ("E" in board_1D):
        break
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   E  E  E
1   E  E  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   E  E  E
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
```

```
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  O  E
It's O's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   O  E  O
1   E  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   O  O  O
1   E  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  O  O
1   O  O  O
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 20
```

Lepše bi bilo, če bi ta del kode shranili v funkcijo, ki nam preveri ali je še mogoča poteza ali ne in nam preprosto vrne `True` ali `False`.

Returning a Value Vsaka funkcija tudi vrne določeno vrednost.

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

[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
```

[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.

[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.

[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
```

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.

[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, 38700661463]}
funkcija(data)

Output:
[43033, 50768369805]
```

Vizualizacija kode

[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.

[100]:
```
def hello(name):
    return f'My name is {name}'
```

[101]:
```
print(hello("Gregor"))
```

```
My name is Gregor
```

[102]:
```
funkcija = hello
print(funkcija("Gregor"))
print(funkcija)
print(type(funkcija))
```

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

[103]:
```
func = [hello, 2 ,3, 'Janez']
print(func[0](func[3]))
```

```
My name is Janez
```

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]
```

Vizualizacija kode

[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]
```

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]
```

Vizualizacija kode

[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]
```

---

7 Križci Krožci

Naloga:

Ustvarite novo fukncije `are_moves_left()`, ki preveri ali so še možne poteze ali ne. Če obstajajo možne poteze funkcija vrne `True`, v nasprotnem primeru vrne `False`.

S pomočjo te vrnjene vrednosti ustavite izvajanje while zanke, če ni več možnih potez. V kolikor ni več možnih potez izpišite končno stanje plošče in `NEODLOČENO`.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   O  E  E
1   E  O  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   O  O  E
1   E  O  E
```

```
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  E  O
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 20
R\C 0  1  2
0   O  O  O
1   E  O  E
2   O  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  O  O
1   O  O  E
2   O  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   O  O  O
1   O  O  O
2   O  O  O
NEODLOČENO
```

Vizualizacija kode

[36]:
```
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]

def display_board():
    print("R\\C 0  1  2")
```

```
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

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

def are_moves_left():
    board_1D = []
    for row in board:
        board_1D.extend(row)
    if not ("E" in board_1D):
        return False
    return True


while True:
    # 1. Prikažemo igralno ploščo
    display_board()

    # 2. Od igralca zahtevamo naj naredi svojo potezo
    make_move("O")

    # 3. Preverimo ali je kdo zmagal

    # 3.a Če je kdo zmagal, izpišemo zmagovalca

    # 3.b Če ni zmagovalca gremo na korak 1
    if not are_moves_left():
        display_board()
        print("NEODLOČENO")
        break
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  E  E
```

```
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   O  E  E
1   E  O  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   O  O  E
1   E  O  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  E  O
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   O  O  O
1   E  O  E
2   E  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 20
R\C 0  1  2
0   O  O  O
1   E  O  E
2   O  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  O  O
1   O  O  E
2   O  O  O
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   O  O  O
1   O  O  O
```

```
2   O  O  O
NEODLOČENO
```

---

8 Križci Krožci

Naloga:

Dodajte še izmenično menjavanje igralca na potezi. Začne `X` nato je na vrsti `O` in tako do konca igre.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   X  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   X  O  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   X  O  X
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   X  O  X
1   O  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   X  O  X
1   O  X  E
2   E  E  E
It's O's turn.
```

```
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   X  O  X
1   O  X  O
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 20
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  O  E
It's X's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  O  X
NEODLOČENO
```

Vizualizacija kode

[37]:
```python
def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

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

def are_moves_left():
    board_1D = []
    for row in board:
        board_1D.extend(row)
    if not ("E" in board_1D):
        return False
    return True
```

```python
# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]
znak = "X"

while True:
    # 1. Prikažemo igralno ploščo
    display_board()

    # 2. Od igralca zahtevamo naj naredi svojo potezo
    make_move(znak)
    znak = "X" if znak == "O" else "O"

    # 3. Preverimo ali je kdo zmagal

    # 3.a Če je kdo zmagal, izpišemo zmagovalca

    # 3.b Če ni zmagovalca gremo na korak 1
    if not are_moves_left():
        display_board()
        print("NEODLOČENO")
        break
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   X  E  E
1   E  E  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 01
R\C 0  1  2
0   X  O  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   X  O  X
1   E  E  E
```

```
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   X  O  X
1   O  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   X  O  X
1   O  X  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 12
R\C 0  1  2
0   X  O  X
1   O  X  O
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 20
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 21
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  O  E
It's X's turn.
Make a move (R/C) (exp: 12): 22
R\C 0  1  2
0   X  O  X
1   O  X  O
2   X  O  X
NEODLOČENO
```

---

9 Križci Krožci

Naloga:

Za konec dodajmo še preverjanje zmagovalca.

Ustvarite novo funkcijo `is_game_over()`, ki preveri ali je kdo od igralcev zmagal. Funkcija naj vrne znak zmagovalca.

Če smo dobili zmagovalca naj se izpiše, kdo je zmagovalec in igra naj se zaključi.

Primeri:

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   E  E  E
1   E  X  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  X  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  E  E
1   X  X  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   O  E  O
1   X  X  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 12
Konec igre:
R\C 0  1  2
0   O  E  O
1   X  X  X
2   E  E  E
ZMAGOVALEC: X
```

Vizualizacija kode

[38]:
```
def display_board():
    print("R\\C 0  1  2")
    for i, row in enumerate(board):
        print(f"{i}   {row[0]}  {row[1]}  {row[2]}")

def make_move(znak):
```

```
    print(f"It's {znak}'s turn.")
    move = input(f"Make a move (R/C) (exp: 12): ")
    row = int(move[0])
    col = int(move[1])
    board[row][col] = znak

def are_moves_left():
    board_1D = []
    for row in board:
        board_1D.extend(row)
    if not ("E" in board_1D):
        return False
    return True

def is_game_over():
    for row in board:
        if row[0] != "E":
            if row[0] == row[1] and row[0] == row[2]:
                return row[0]

    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 board[0][i]

    if board[0][0] != "E":
        if board[0][0] == board[1][1] and board[0][0] == board[2][2]:
            return board[0][0]

    if board[0][2] != "E":
        if board[0][2] == board[1][1] and board[0][2] == board[2][0]:
            return board[0][2]

    return False


# Ustvarimo spremenljivko v kateri bomo hranili stanje naše plošče
board = [["E", "E", "E"],
         ["E", "E", "E"],
         ["E", "E", "E"]]
znak = "X"

while True:
    # 1. Prikažemo igralno ploščo
    display_board()

    # 2. Od igralca zahtevamo naj naredi svojo potezo
```

```
    make_move(znak)
    znak = "X" if znak == "O" else "O"

    # 3. Preverimo ali je kdo zmagal
    winner = is_game_over()
    if winner:
        print("Konec igre:")
        display_board()
        # 3.a Če je kdo zmagal, izpišemo zmagovalca
        print(f"ZMAGOVALEC: {winner}")
        break

    # 3.b Če ni zmagovalca gremo na korak 1
    if not are_moves_left():
        display_board()
        print("NEODLOČENO")
        break
```

```
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 11
R\C 0  1  2
0   E  E  E
1   E  X  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 00
R\C 0  1  2
0   O  E  E
1   E  X  E
2   E  E  E
It's X's turn.
Make a move (R/C) (exp: 12): 10
R\C 0  1  2
0   O  E  E
1   X  X  E
2   E  E  E
It's O's turn.
Make a move (R/C) (exp: 12): 02
R\C 0  1  2
0   O  E  O
1   X  X  E
2   E  E  E
It's X's turn.
```

```
Make a move (R/C) (exp: 12): 12
Konec igre:
R\C 0  1  2
0   O  E  O
1   X  X  X
2   E  E  E
ZMAGOVALEC: X
```

Pri funkcijah lahko omenimo še **variable scope**.

10 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.

[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.

[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.

[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.

[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.

[74]:
```
def funkcija(l):
    print(l)
    l[0] = 100

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

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

[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}
```

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

[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.

[77]:
```
def funkcija():
    global spr1
    spr1 = 5
    print(f"Spr1: {spr1}")

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

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

[ ]:

[ ]:

[ ]: