P3_Prenovljeno_Funkcije-Copy1

May 15, 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!
```

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.

[2]:
```
# 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

[4]:
```
# 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): 00
R\C 0  1  2
0   X  E  E
1   E  E  E
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
```

[5]:
```
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.

[6]:
```
print("Začetek programa")
hello2()
print("Nadaljevanje programa")

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

```
Začetek programa
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/2897170123.py in <module>
      1 print("Začetek programa")
----> 2 hello2()
      3 print("Nadaljevanje programa")
      4
      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
```

[8]:
```
# Rešitev
def fun():
    emšo = input("Vnesi emšo: ")

    letnica = int(emšo[4:7]) + 1000
    print(f"Star si {2023-letnica} let")
```

```
fun()
```

```
Vnesi emšo: 0102999500111
Star si 24 let
```

2.2 The pass statements

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

[9]:
```
if True:

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

```
  File "C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/3295335100.py", lin
↪3
    print("Hello") # should give IndentationError
    ^
IndentationError: expected an indented block
```

[10]:
```
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

[11]:
```
# 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): 11
R\C 0  1  2
0   E  E  E
1   E  X  E
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

[12]:
```
# 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): 22
R\C 0  1  2
0   E  E  E
1   E  E  E
2   E  E  X
```

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

[13]:
```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!

[14]:
```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.

[15]:
```
# 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)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/426141945.py in <module>
      5     print(f"Z vrednost: {z}")
      6
----> 7 funkcija_1(1, 2)

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

[16]:
```
# 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)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/28376062.py in <module>
      5     print(f"Z vrednost: {z}")
      6
----> 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
```

[17]:
```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")
```

[18]:
```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.

[19]:
```
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)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/2810305550.py 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.

[20]:
```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

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

```
Pozdravljeni gospod Miha Novak.
```

[21]:
```
def pozdrav(naslavljanje, ime, priimek):
    print(f"Pozdravljeni {naslavljanje} {ime} {priimek}.")

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

```
  File "C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/3349422776.py", lin
↪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
```

[22]:
```
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.

[23]:
```
def funkcija(x, y, z=0):
    print(x + y + z)

funkcija(1, 2)
```

```
3
```

[24]:
```
def funkcija(x, y=0, z):
    print(x + y + z)

funkcija(1, 2, 3)
```

```
  File "C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_8920/3799133569.py", lin
↪1
    def funkcija(x, y=0, z):
                         ^
SyntaxError: non-default argument follows default argument
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: