08_Napake_in_Importiranje

April 19, 2022

1 Try and except statements

Če za neki del kode predvidimo da bi lahko prišlo do napake, uporabimo try-except.

Ko se zgodi napaka Python preveri, če je del kode, ki je vrgel napako znotraj try-except. Če je, Python pogleda, če kateri od exceptov poskrbi za to napako. Če najde pravilen except se izvede njegov del kode s katero poskrbimo za napako in program se nadaljuje od konca try-except.

Če Python ne najde pravilnega except-a oziroma vrstica, ki je vrgla napako ni znotraj try-excepta, gre Python 1 nivo višje v kodi. Se pravi, če je bila napaka storjena znotraj funkcije in za napako nismo poskrbeli, gre Python nazaj na vrstico, ki je klicala to funkcijo, in ta vrstica bo tretirana, kot da je ona vrgla napako.

Kaj so prednosti exception handling: * Exception se lahko preprosto posreduje naprej, dokler ne naletimo na del kode, ki lahko poskrbi za ta exception * Exceptions imajo veliko built-in uporabnih error informacij (lahko dobimo traceback, ki nam pomaga pri debuggiranju)

> Here are a few other advantages of exception handling: * It separates normal code from code that handles errors. * Exceptions can easily be passed along functions in the stack until they reach a function which knows how to handle them. * The intermediate functions don't need to have any error-handling code. * Exceptions come with lots of useful error information built in – for example, they can print a traceback which helps us to see exactly where the error occurred.

[1]:
```
def fun1(a,b):
    return a/b
def fun2(a,b):
    return fun1(a,b)

print(fun2(1,0))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/202410779.py in <module>
      4     return fun1(a,b)
      5
----> 6 print(fun2(1,0))

C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/202410779.py in fun2(a, b)
      2     return a/b
```

```
      3 def fun2(a,b):
----> 4     return fun1(a,b)
      5
      6 print(fun2(1,0))

C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/202410779.py in fun1(a, b)
      1 def fun1(a,b):
----> 2     return a/b
      3 def fun2(a,b):
      4     return fun1(a,b)
      5

ZeroDivisionError: division by zero
```

[2]:
```
try:
    print("Izvedi ta print")
except:
    print("Prišlo je do napake")
```

```
Izvedi ta print
```

[3]:
```
try:
    print("Nekaj posebnega")
    #a = 1/0
    b = neka_spremenljivka
    print("Po napaki")
except ZeroDivisionError as err:
    print("Ne smeš deljiti z 0!")
    print(type(err))
    print(f"Prišlo je do napake: {err}")
except NameError as er:
    print(f"Prišlo je do NameError-ja <{er}>")


print("Dodatni print")
```

```
Nekaj posebnega
Prišlo je do NameError-ja <name 'neka_spremenljivka' is not defined>
Dodatni print
```

Dodamo lahko tudi `else:` in pa `finally:`, vendar sta opcijska (ju lahko izpustimo). Uporabimo po potrebi.

Ponavadi se `finally` uporabi za čiščenje kode.

[4]:
```
try:
    pass
except:
```

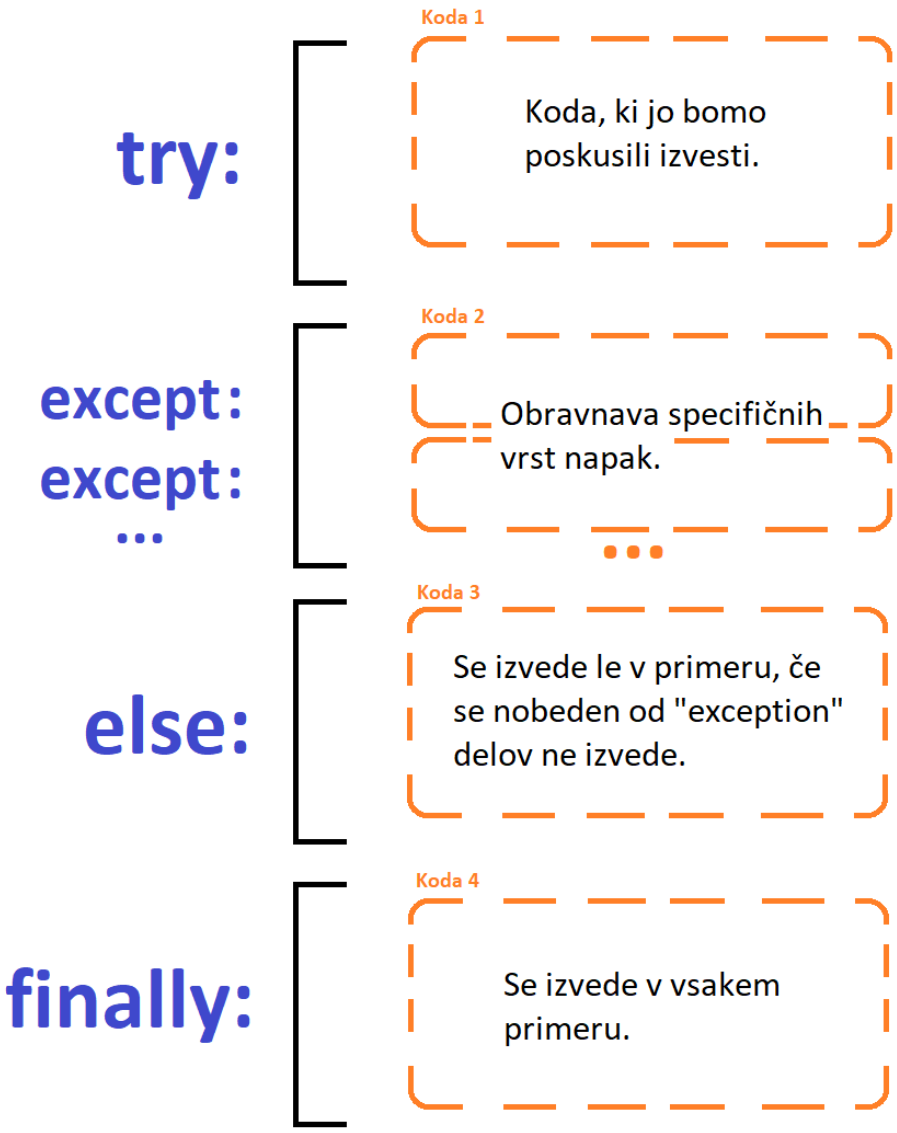
```
    pass
else:
    pass
finally:
    pass
```

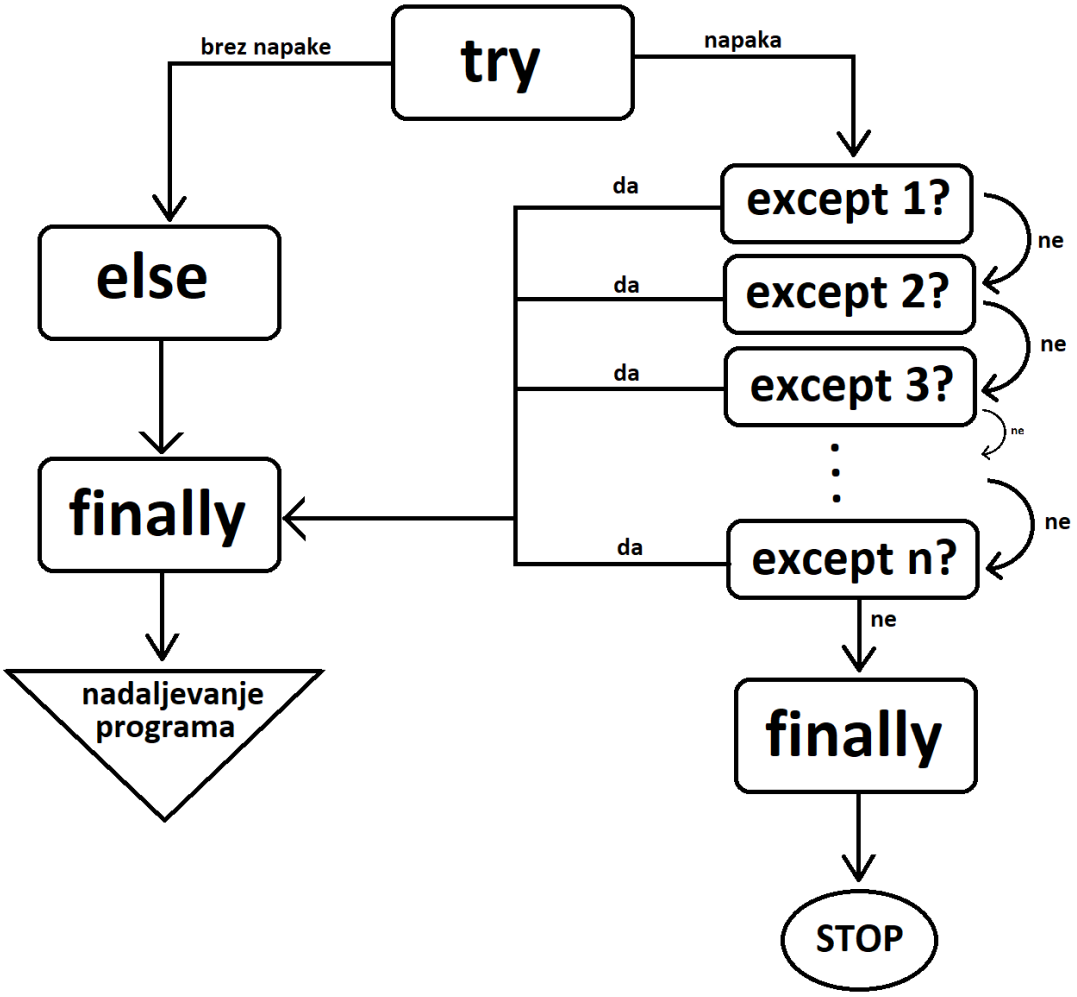
[5]:
```
try:
    a = 1/0
except ValueError as e:
    print(f"ValueError: {e}")
#except ZeroDivisionError as e:
#    print(f"ZeroDivisionError: {e}")
else:
    print("Vse se je izvedlo pravilno!")
finally:
    print("Čiščenje kode!!!!")
print("sdfsdgffd")
```

```
Čiščenje kode!!!!
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/2766774547.py in <module>
      1 try:
----> 2     a = 1/0
      3 except ValueError as e:
      4     print(f"ValueError: {e}")
      5 #except ZeroDivisionError as e:

ZeroDivisionError: division by zero
```


Koda 1
try:
Koda, ki jo bomo poskusili izvesti.
Koda 2
except:
except:
...
Obravnava specifičnih vrst napak.
Koda 3
else:
Se izvede le v primeru, če se nobeden od "exception" delov ne izvede.
Koda 4
finally:
Se izvede v vsakem primeru.



1.0.1 Minimalna uporaba

Poleg try je potrbno uporabiti vsaj except ali pa finally!!!

[6]:
```
try:
    a = 1/0
except:
    print("Prišlo je do napake")
```

```
Prišlo je do napake
```

[7]:
```
try:
    a = 1/0
finally:
    print("Izvede se finally")
```

```
Izvede se finally
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/2598489934.py in <module>
      1 try:
----> 2     a = 1/0
      3 finally:
      4     print("Izvede se finally")

ZeroDivisionError: division by zero
```

[8]:
```
try:
    print("try")
    print(1/0)
except:
    print("exception")
#else:
#    print("else")
#finally:
#    print("finally")
```

```
try
exception
```

1.0.2 Napake znotraj except/else/finally

[9]:
```
# error znotraj finally statementa
try:
    g = 1/0
except ZeroDivisionError:
    print("Deljenje z 0 ni dovoljeno.")
else:
    print("Nič ni bilo narobe.")
finally:
    print("Pravilno/Napačno, vseeno?")
    print(g)
print("Try-except je deloval perfektno!")
```

```
Deljenje z 0 ni dovoljeno.
Pravilno/Napačno, vseeno?
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/1774436082.py in <module>
      8 finally:
      9     print("Pravilno/Napačno, vseeno?")
---> 10     print(g)
```

```
     11 print("Try-except je deloval perfektno!")

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

1.0.3 Uporaba Except objekta

Python Exceptions imajo tudi ostale informacije, kot je sporočilo, ki pove zakaj je prišlo do napake.

Do teh podatkov lahko dostopamo na sledeč način:

[10]:
```
try:
    a = int(input("a: "))
    b = int(input("b: "))
    print(a/b)
except Exception as e:
    print("Neuporabni Except" + str(e))
except ZeroDivisionError as e:
    print(f"Prišlo je do napake: {e}")
except ValueError as e:
    print(f"Prišlo je do napake: {e}")
else:
    print("Pravilno izvedena try koda.")
finally:
    print("Pravilno/nepravilno, vseeno!")
```

```
a:  1
b:  0

Neuporabni Exceptdivision by zero
Pravilno/nepravilno, vseeno!
```

[11]:
```
def funkcija(n):
    return 1/n

try:
    funkcija(0)
except ArithmeticError as e:
    print('Napaka '+str(e))# pokaži da "e" ni string
    print(type(e))
print('End')
```

```
Napaka division by zero
<class 'ZeroDivisionError'>
End
```

V našem except delu smo specificirali, da bomo pohendlali specifično `ArithmeticError`, če nastane. Lahko bi rekli, da bomo pohendlali kakoršenkoli error ampak potem ne vemo kakšna napaka se je zgodila. Oziroma, mogoče hočemo drugačen error pohendlati na drugačen način.

[12]:
```
try:
    x = int(input("Vnesi x: "))
    y = int(input("Vnesi y: "))
    print(f'{x} / {y} = {x/y}')
except(ValueError, ZeroDivisionError) as e:
    print("Nekaj ni vredu.")
    print(e)
```

```
Vnesi x:  safsafd
```

```
Nekaj ni vredu.
invalid literal for int() with base 10: 'safsafd'
```

[13]:
```
try:
    x = int(input("Vnesi x: "))
    y = int(input("Vnesi y: "))
    print(f'{x} / {y} = {x/y}')
except ValueError:
    print("Obe spremeljivki morata biti številki.")
except ZeroDivisionError:
    print("Deljitelj ne sme biti 0.")
```

```
Vnesi x:  5
Vnesi y:  0
```

```
Deljitelj ne sme biti 0.
```

V primeru napake bo Python preveril vsak `except` od vrha navzdol, če se tipa napaki ujemata. Če se napaka ne ujema z nobenim `except` potem se bo program ustavil z napako.

Če se ujemata bo pa `except` obravnaval error. `Except` sprejme errorje tega razreda in vse, ki dedujejo iz tega razreda.

(https://docs.python.org/3/library/exceptions.html)

Se pravi, če damo kot prvi `except except Exception` bomo z njim prestregl vse, ker vsi dedujejo iz tega classa.

BaseException * SystemExit * KeyboardInterrupt * GeneratorExit * Exception * * StopIteration * * StopAsyncIteration * * ArithmeticError * * * FloatingPointError * * * OverflowError * * * ZeroDivisionError * * AssertionError * * AttributeError * * BufferError * * EOFError * * ImportError * * * ModuleNotFoundError * * LookupError * * * IndexError * * * KeyError * * MemoryError * * NameError * * * UnboundLocalError * * OSError * * * BlockingIOError * * * ChildProcessError * * * ConnectionError * * * * BrokenPipeError * * * * ConnectionAbortedError * * * * ConnectionRefusedError * * * * ConnectionResetError * * * FileExistsError * * * FileNotFoundError * * * InterruptedError * * * IsADirectoryError * * * NotADirectoryError * * * PermissionError * * * ProcessLookupError * * * TimeoutError * * ReferenceError * * RuntimeError * * * NotImplementedError * * * RecursionError * * SyntaxError * * * IndentationError * * * * TabError * * SystemError * * TypeError * * ValueError * * * UnicodeError * * * * UnicodeDecodeError * * * * UnicodeEncodeError * * * * UnicodeTranslateError * * Warning * * * DeprecationWarning * * * PendingDeprecationWarning * * * RuntimeWarning * * * SyntaxWarning * * * UserWarn-

ing * * * FutureWarning * * * ImportWarning * * * UnicodeWarning * * * BytesWarning * * * ResourceWarning

[14]:
```
import inspect

# Primer: Različno hendlanje različnih errorjev.
try:
    x = int(input("Vnesi x: "))
    y = int(input("Vnesi y: "))
    print(f'{x} / {y} = {x/y}')
except Exception:
    print("Zmeraj ta prestreže.")
except ValueError:
    print("Obe spremeljivki morata biti številki.")
except ZeroDivisionError:
    print("Deljitelj ne sme biti 0.")


print(inspect.getmro(Exception))
print(inspect.getmro(ValueError))
print(inspect.getmro(ZeroDivisionError))
```

```
Vnesi x:  10
Vnesi y:  0
```

```
Zmeraj ta prestreže.
(<class 'Exception'>, <class 'BaseException'>, <class 'object'>)
(<class 'ValueError'>, <class 'Exception'>, <class 'BaseException'>, <class
'object'>)
(<class 'ZeroDivisionError'>, <class 'ArithmeticError'>, <class 'Exception'>,
<class 'BaseException'>, <class 'object'>)
```

2 NALOGA

Aplikacija ki sprejme ime + starost `input()` in izpiše nek stavek.

Na koncu vpraša ali se program ponovi! [y/n] `input()`

[15]:
```
ime = "aanze"
starost = "sdfsdf"
print(f"Uporabniku je ime {ime} in star je {starost} let.")
```

```
Uporabniku je ime aanze in star je sdfsdf let.
```

[16]:
```
try:
    ime = input('Ime')
    starost = int(input('Vnesi starost: '))
except ValueError as e:
```

```
    print('Pozor, iz vnosa {starost} starosti nisem mogel dolociti. Uporabil␣
 ↪sem privzeto vrednost.')
    starost = 25
finally:
    print(f"Uporabniku je ime {ime} in star je {starost} let.")
```

```
Ime assaf
Vnesi starost:  asdsad

Pozor, iz vnosa {starost} starosti nisem mogel dolociti. Uporabil sem privzeto
vrednost.
Uporabniku je ime assaf in star je 25 let.
```

[17]:
```
while True:
    try:
        ime = str(input("Vpiši ime uporabnika: "))
        starost = int(input("Vpiši starost uporabnika: "))
    except ValueError as e:
        print("Starost ni bila podana v pravilni obliki. Ponovi vnos ...")
    else:
        print(f"Uporabniku je ime {ime} in star je {starost} let.")
    finally:
        nadaljevanje = input("Želiš vnesti novega uporabnika [y/n]? ")
        if nadaljevanje == "y":
            pass
        elif nadaljevanje == "n":
            print("Program se zaključuje.")
            break
        else:
            print("Tvoja odločitev ni jasno izražena. Poskusi znova.")
```

```
Vpiši ime uporabnika:
Vpiši starost uporabnika:  -8

Uporabniku je ime  in star je -8 let.

Želiš vnesti novega uporabnika [y/n]?  y
Vpiši ime uporabnika:  anze
Vpiši starost uporabnika:  2

Uporabniku je ime anze in star je 2 let.

Želiš vnesti novega uporabnika [y/n]?  n

Program se zaključuje.
```

[18]:
```
try:
    while True:
        ime = input("Vnesi ime: ")
        starost = int(input("Vnesi starost:"))
```

```
        print(f"Uporabniku je ime {ime} in star je {starost} let.")

        ponovitev = input("Ali naj se program ponovi ( y / n ) ? ")

        if ponovitev == "n":
            break


except ValueError as e:
    print(f"Napačno starost {e}")
```

```
Vnesi ime:  sandslovn
Vnesi starost: -9

Uporabniku je ime sandslovn in star je -9 let.

Ali naj se program ponovi ( y / n ) ?  y
Vnesi ime:  8
Vnesi starost: 8

Uporabniku je ime 8 in star je 8 let.

Ali naj se program ponovi ( y / n ) ?  n
```

2.0.1 Raise

`Raise` se uporabljamo, da lahko tudi sami pokličemo napako.

[21]:
```
class ImeNiPravilno(Exception): # custom Exception
    pass
```

[22]:
```
while True:
    try:
        ime = str(input("Vpiši ime uporabnika: "))
        if ime[0]:
            raise ImeNiPravilno("Ime ni bilo vnešeno pravilno!")
        starost = int(input("Vpiši starost uporabnika: "))
        if starost < 0:
            raise ValueError("Starost ne sme biti negativna!!!")
    except ValueError as e:
        print("Starost ni bila podana v pravilni obliki. Ponovi vnos ...")
        print(f"ValueError: {e}")

    except ImeNiPravilno as e:
        print(f"Napaka: {e}")

    else:
        print(f"Uporabniku je ime {ime} in star je {starost} let.")
    finally:
        nadaljevanje = input("Želiš vnesti novega uporabnika [y/n]? ")
```

```
        if nadaljevanje == "y":
            pass
        elif nadaljevanje == "n":
            print("Program se zaključuje.")
            break
        else:
            print("Tvoja odločitev ni jasno izražena. Poskusi znova.")
```

```
Vpiši ime uporabnika:
Želiš vnesti novega uporabnika [y/n]?

Tvoja odločitev ni jasno izražena. Poskusi znova.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/1712204161.py in <module>
      2     try:
      3         ime = str(input("Vpiši ime uporabnika: "))
----> 4         if ime[0]:
      5             raise ImeNiPravilno("Ime ni bilo vnešeno pravilno!")
      6         starost = int(input("Vpiši starost uporabnika: "))

IndexError: string index out of range
```

V tem primeru smo sami `raise`-ali `ValueError` v primeru, ko je bil `input()` pravilen ampak je bil negativen.

Uporabili smo `ValueError` z našim sporočilom. Lahko bi uporabili tudi katerokoli drugo napako.

2.0.2 Pisanje svojih Exceptionov

[23]:
```
class B(Exception):
    pass

class C(B):
    pass

class D(B):
    pass

class E(D):
    pass

for error in [E, D, C, B]:
    try:
        raise error
    except D: # treba pazt zaporedje exceptov!
        print("D")
```

```
    except C:
        print("C")
    except B:
        print("B")
```

```
D
D
C
B
```

[24]:
```
class ImeNiPravilno(Exception):
    pass
```

[25]:
```
while True:
    try:
        ime = str(input("Vpiši ime uporabnika: "))
        if len(ime) < 1:
            raise ImeNiPravilno("Ime ni bilo vnešeno pravilno!")
        starost = int(input("Vpiši starost uporabnika: "))
        if starost < 0:
            raise ValueError("Starost ne sme biti negativna!!!")
    except ValueError as e:
        print("Starost ni bila podana v pravilni obliki. Ponovi vnos ...")
        print(f"ValueError: {e}")

    except ImeNiPravilno as e:
        print(f"Napaka: {e}")

    else:
        print(f"Uporabniku je ime {ime} in star je {starost} let.")
    finally:
        nadaljevanje = input("Želiš vnesti novega uporabnika [y/n]? ")
        if nadaljevanje == "y":
            pass
        elif nadaljevanje == "n":
            print("Program se zaključuje.")
            break
        else:
            print("Tvoja odločitev ni jasno izražena. Poskusi znova.")
```

```
Vpiši ime uporabnika:  a
Vpiši starost uporabnika:  a

Starost ni bila podana v pravilni obliki. Ponovi vnos …
ValueError: invalid literal for int() with base 10: 'a'

Želiš vnesti novega uporabnika [y/n]?  y
Vpiši ime uporabnika:  sasadf
```

```
Vpiši starost uporabnika:  -9

Starost ni bila podana v pravilni obliki. Ponovi vnos …
ValueError: Starost ne sme biti negativna!!!

Želiš vnesti novega uporabnika [y/n]?  y
Vpiši ime uporabnika:  anze
Vpiši starost uporabnika:  8

Uporabniku je ime anze in star je 8 let.

Želiš vnesti novega uporabnika [y/n]?  n

Program se zaključuje.
```

2.0.3 Assertion

`assert Expression[, Arguments]`

`assert` se uporablja za preverbo vrednosti programa oziroma, da je neka vrednost programa v skladu s pričakovanji.

..deluje podobno kot `raise`, vendar s tem da preveri neko vrednost in raise-a `AssertionError`

Načeloma se uporablja kot pomoč pri hitremu debug-iranju programa.

[26]:
```
a = 20
assert a < 10, "Spremenljivka <a> naj bi bila manjša od 10 !!!"
print("sdfdsf")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
C:\Users\FAKULT~1\AppData\Local\Temp/ipykernel_9412/825020590.py in <module>
      1 a = 20
----> 2 assert a < 10, "Spremenljivka <a> naj bi bila manjša od 10 !!!"
      3 print("sdfdsf")

AssertionError: Spremenljivka <a> naj bi bila manjša od 10 !!!
```

[ ]:
```

```

3 Importing

`Importing` je način, kako lahko kodo iz ene datoteke/modula/package uporabimo v drugi datoteki/modulu. * **module** je datoteka, ki ima končnico *.py* * **package** je direktorij, ki vsebuje vsaj en modul

Da importiramo modul uporabimo besedo `import`.

`import moj_modul`

Python sedaj prvo preveri ali se *moj_modul* nahaja v **sys.modules** - to je dictionary, ki hrani imena vseh importiranih modulov.

Če ne najde imena, bo nadaljeval iskanje v `built-in` modulih. To so moduli, ki pridejo skupaj z inštalacijo Pythona. Najdemo jih lahko v Python Standardni Knjižnjici - https://docs.python.org/3/library/ .

Če ponovno ne najde našega modula, Python nadaljuje iskanje v **sys.path** - to je list direktorijev med katerimi je tudi naša mapa.

Če Python ne najde imena vrže **ModuleNotFoundError**. V primeru, da najde ime, lahko modul sedaj uporabljamo v naši datoteki.

Za začetek bomo importiral **math** built-in modul, ki nam omogoča naprednejše matematične operacije, kot je uporaba korenjenja.

math documentation - https://docs.python.org/3/library/math.html

Da pogledamo katere spremenljivke / funkcije / objekti / itd. so dostopni v naši kodi lahko uporabimo **dir()** funkcijo.

dir documentation - https://docs.python.org/3/library/functions.html#dir

[27]:
```
import math

moja_spremenljivka = 5
print(dir())

print(moja_spremenljivka)
print(math)
```

```
['B', 'C', 'D', 'E', 'ImeNiPravilno', 'In', 'Out', '_', '__', '___',
'__builtin__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__', '_dh', '_i', '_i1', '_i10', '_i11', '_i12', '_i13',
'_i14', '_i15', '_i16', '_i17', '_i18', '_i19', '_i2', '_i20', '_i21', '_i22',
'_i23', '_i24', '_i25', '_i26', '_i27', '_i3', '_i4', '_i5', '_i6', '_i7',
'_i8', '_i9', '_ih', '_ii', '_iii', '_oh', 'a', 'b', 'error', 'exit', 'fun1',
'fun2', 'funkcija', 'get_ipython', 'ime', 'inspect', 'math',
'moja_spremenljivka', 'nadaljevanje', 'ponovitev', 'quit', 'starost', 'x', 'y']
5
<module 'math' (built-in)>
```

S pomočjo **dir(...)** lahko tudi preverimo katere spremenljivke, funkcije, itd. se nahajajo v importiranih modulih.

[28]:
```
import math

moja_spremenljivka = 5
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign',
'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi',
'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc', 'ulp']
```

Funkcijo, spremenljivko, atribut v math modulu uporabimo na sledeč način:

[29]:
```
import math

print(math.sqrt(36))
```

```
6.0
```

Naloga:

S pomočjo math modula izračunajte logaritem 144 z osnovo 12.

https://docs.python.org/3/library/math.html

[30]:
```
# Rešitev
import math

math.log(144, 12)
```

[30]:
```
2.0
```

[ ]:
```

```

4 Importing our own module

Ustvarimo novo datoteko **moj_modul.py** zraven naše datoteke s kodo.

4.0.1 moj_modul.py

[31]:
```
class Pes():
    def __init__(self, ime):
        self.ime = ime


def sestevalnik(a, b):
    return a+b


moja_spremenljivka = 100
```

4.0.2 skripta.py

```
[ ]: import moj_modul

print(dir())

fido = moj_modul.Pes("fido")
print(fido.ime)

print(moj_modul.sestevalnik(5, 6))

print(moj_modul.moja_spremenljivka)
```

5 Načini importiranja

Importiramo lahko celotno kodo ali pa samo specifične funkcije, spremenljivke, objekte, itd.

Celotno kodo importiramo na sledeči način:

`import moj_modul`

```
[ ]: import moj_modul

print(dir())

fido = moj_modul.Pes("fido")
print(fido.ime)

print(moj_modul.sestevalnik(5, 6))

print(moj_modul.moja_spremenljivka)
```

Specifične zadeve importiramo na sledeč način:

`from moj_modul import moja_spremenljivka`

```
[ ]: from moj_modul import moja_spremenljivka

print(dir())
print(moja_spremenljivka)
```

```
[ ]: from moj_modul import sestevalnik

print(dir())
print(sestevalnik(5,6))
```

```
[ ]: from moj_modul import Pes

print(dir())
```

```
fido = Pes("fido")
print(fido.ime)
```

Importirane zadeve se lahko shrani tudi pod drugim imenom

`import moj_modul as mm`

[ ]:
```
import moj_modul as mm

print(dir())

fido = mm.Pes("fido")
print(fido.ime)

print(mm.sestevalnik(5, 6))

print(mm.moja_spremenljivka)
```

[ ]:
```
from moj_modul import sestevalnik as sum_

print(dir())
print(sum_(5,6))
```

Za premikanje med direktoriji med importiranjem se uporabja " **.** " .

`from package1.module1 import function1`

5.0.1 modul2.py

[ ]:
```
def potenciranje(x, y):
    return x**y

spremenljivka2 = 200
```

5.0.2 skripty.py

[ ]:
```
from moj_package import modul2

print(dir())

print(modul2.potenciranje(2,3))
```

[ ]:
```
from moj_package.modul2 import potenciranje

print(dir())
```

```
print(potenciranje(2,3))
```

Naloga:

Ustvarite nov modul imenovan naloga1.py. Znotraj modula napišite funkcijo pretvornik(x, mode), ki spreminja radiane v stopinje in obratno.

Funkcija naj sprejme 2 argumenta. Prvi argument je vrednost, katero želimo pretvoriti. Drugi argument, imenovan mode pa nam pove v katero enoto spreminjamo.

`mode = "deg2rad" pomeni, da spreminjamo iz stopinj v radiane`
`mode = "rad2deg" pomeni, da spreminjamo iz radianov v stopinje`

Za pomoč pri pretvarjanju uporabite math modul.

Zravn modula prilepite podano skripto test.py in to skripto zaženite.

[ ]:
```
# test.py
import naloga1


r1 = naloga1.pretvornik(180, mode="deg2rad")
if float(str(r1)[:4]) == 3.14:
    print("Rešitev pravilna.")
else:
    print("Nekaj je narobe.")

r2 = naloga1.pretvornik(360, mode="deg2rad")
if float(str(r2)[:4]) == 6.28:
    print("Rešitev pravilna.")
else:
    print("Nekaj je narobe.")

r3 = naloga1.pretvornik(1.5707963267948966, mode="rad2deg")
if r3 == 90:
    print("Rešitev pravilna.")
else:
    print("Nekaj je narobe.")

r3 = naloga1.pretvornik(4.71238898038469, mode="rad2deg")
if r3 == 270:
    print("Rešitev pravilna.")
else:
    print("Nekaj je narobe.")
```

[1]:
```
# Rešitev
import math

def pretvornik(x ,mode="deg2rad"):
    if mode == "deg2rad":
```

```
        return math.radians(x)
    elif mode == "rad2deg":
        return math.degrees(x)
```

[ ]:
```

```

Importiramo lahko tudi vse naenkrat z uporabo " * " vendar se to odsvetuje, saj nevem kaj vse smo importirali in lahko na tak način ponesreči kaj spremenimo.

[ ]:
```
from math import *

print(dir())
print(pi)

pi = 3
print(pi)
```

5.0.3 ___name___ variable

Python ima posebno spremenljivko ___name___. Spremenljivka dobi vrednost, glede na to kako smo zagnali naš modul.

Če zaženemo naš modul direktno, bo spremenljivka enaka ___main___.

m1.py

[ ]:
```
def my_name():
    print(__name__)

my_name()
```

To bi delovalo v primeru, ko smo ustvarili svoj modul in vanj sproti zapisali kakšen preprost test naše kode.

Problem se pojavi, ko moj_modul importiramo, sam se ob importiranju celotna koda v modulu izvede.

[2]:
```
import m1

print(__name__)
print(m1.__name__)
```

```
__main__
m1
```

Da preprečimo nepotrebno izvajanje funkcij lahko uporabimo ___name___spremenljivko.

Naš modul bi sedaj izgledal sledeče:

m1.py

[ ]:
```
def my_name():
    print(__name__)

if __name__ == "__main__":
    my_name()
```

skripta.py

[ ]:
```
import m1

print(__name__)
print(m1.__name__)
```

[ ]:
```

```