Banka

Napišite program:

```
class Oseba():
    def __init__(self, ime, priimek):
        # za to instanco naj ustvari spremenljivki ime in priimek

    def opis(self):
        # vrne naj string, znotraj katerega imamo ime in priimek


class Stranka(): # class naj deduje od razreda Oseba()
    def nastavi_stanje(self, stanje):
        # metoda naj ustvari spremenljivko samo za to instaco razreda. V
rednost naj bo "stanje" oziroma default vrednost naj bo 0. Metoda naj na
to vrne vrednost spremenljivke stanje

    def dvig(self, znesek):
        # Od stanja naj se odšteje znesek.
        # V kolikor ni dovolj denarja na računu naj se dvigne z banke ce
lotno stanje
        # Na koncu naj metoda vrne dvignjen znesek

    def polog(self, znesek):
        # metoda naj doda velikost zneska stanju
        # nato naj metoda vrne novo stanje


INPUT:
objekt = Stranka("Gregor", "Balkovec")
print(objekt.opis())
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))

OUTPUT:
Gregor Balkovec
0.0
5000.0
2000
Dal ti bom samo 3000.0
3000.0
```

In [1]:
```
class Oseba():
    def __init__(self, ime, priimek):
        self.ime = ime
        self.priimek = priimek

    def opis(self):
        return self.ime + " " + self.priimek


class Stranka(Oseba):

    def nastavi_stanje(self, stanje=0.0):
        self.stanje = stanje
        return self.stanje

    def dvig(self, znesek):
        dvig = 0
        if znesek > self.stanje:
            print("Dal ti bom samo", self.stanje)
            dvig = self.stanje
            self.stanje = 0
        else:
            self.stanje -= znesek
            dvig = znesek
        return dvig

    def polog(self, znesek):
        self.stanje += znesek
        return self.stanje


objekt = Stranka("Gregor", "Balkovec")
print(objekt.opis())
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))
```

```
Gregor Balkovec
0.0
5000.0
2000
Dal ti bom samo 3000.0
3000.0
```

Ustvarite razred `Polica` .

- Vsaka instanca razreda naj ima:
- - **knjige** -> list naslovov knjig, ki se nahajajo na polici
- - **max_knjig** -> integer vrednost, ki pove koliko knjig, gre maximalno na polico

- Ko ustvarimo instanco razreda vanj posredujemo številko maximalnih knjig na polici.
- Ko ustvarimo instanco razreda vanj posredujemo lahko tudi list naslovov knjig, ki se že nahajajo na polici. Če takega seznama ne posredujemo naj ima polica prazen seznam.
- Razred naj ima metodo `kaj_je_na_polici` , ki naj vrne list naslovov knjig
- Razred naj ima metodo `dodaj_knjigo` , ki kot argument prejme string naslova knjige. To knjigo naj doda v list naslovov knjig, če s tem ne presežemo maximalno število knjig. Če bi presegli to število knjige ne dodamo.
- Razred naj ima metodo `uredi_knjige` , ki kot argument **ascending** prejme boolean vrednost, ki nam pove ali naj bodo knjige urejene (glede na prvo črko) v A->Z (vrednost True) oziroma Z->A (vrednost False). Če ta vrednost ni bila posredovana naj bo default vrstni red A->Z. Metoda naj uredi list naslovov knjig in tega nato vrne

```
polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet", "Kr
autov Strojniški Priročnik", "SSKJ"])
print(polica.kaj_je_na_polici())
==> ['The Witcher', 'Dune', 'Harry Potter', 'Hamlet', 'Krautov Strojnišk
i Priročnik', 'SSKJ']

print(polica.uredi_knjige())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher']

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher', 'Romeo in Julija']

polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher', 'Romeo in Julija']

print(polica.uredi_knjige(False))
==> ['The Witcher', 'SSKJ', 'Romeo in Julija', 'Krautov Strojniški Priro
čnik', 'Harry Potter', 'Hamlet', 'Dune']
```

In [86]:
```
class Polica:
    def __init__(self, max_knjig, knjige=list()):
        self.max_knjig = max_knjig
        self.knjige = knjige

    def kaj_je_na_polici(self):
        return self.knjige

    def dodaj_knjigo(self, knjiga):
        if len(self.knjige) < self.max_knjig:
            self.knjige.append(knjiga)

    def uredi_knjige(self, ascending = True):
        if ascending:
            self.knjige.sort()
        else:
            self.knjige.sort(reverse=True)
        return self.kaj_je_na_polici()



polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet", "Tintin", "S
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige())

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())

polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige(False))
```

```
['The Witcher', 'Dune', 'Harry Potter', 'Hamlet', 'Tintin', 'SSKJ']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin', 'Romeo in J
ulija']
['Dune', 'Hamlet', 'Harry Potter', 'SSKJ', 'The Witcher', 'Tintin', 'Romeo in J
ulija']
['Tintin', 'The Witcher', 'SSKJ', 'Romeo in Julija', 'Harry Potter', 'Hamlet',
'Dune']
```

Errors

Error's so napake v programu, ki nam ponavadi zaustavijo izvajanje programa.

Klasificiramo jih v:

- Snytaks errors
- Runtime errors
- Logical errors

Syntax errors

Syntax errors so napake pri uporabi Python jezika.

Python bo našel te napake med parsanjem našega programa. Če najde takšno napako bo exit-u brez, da bi pognal ta del kode.

Najbolj pogoste Syntax napake so:

- izpustitev keyword
- uporaba keyword na napačnem mestu
- izpustitev simbolov, kot je :
- napačno črkovanje
- napačen indentation

In [1]:
```
# Primer: manjka keyword def
myfunction(x, y):
    return x + y
```

```
  File "<ipython-input-1-8b32d31d1203>", line 2
    myfunction(x, y):
                     ^
SyntaxError: invalid syntax
```

In [2]:
```
else:
    print("Hello!")
```

```
  File "<ipython-input-2-429811f9164b>", line 1
    else:
    ^
SyntaxError: invalid syntax
```

In [3]:
```
# Primer: manjka :
if mark >= 50
    print("You passed!")
```

```
  File "<ipython-input-3-2bfd10af2cba>", line 2
    if mark >= 50
                 ^
SyntaxError: invalid syntax
```

In [4]:
```
# Primer: napačno črkovanje "else"
if arriving:
    print("Hi!")
esle:
    print("Bye!")
```

```
  File "<ipython-input-4-1cca186d8b5e>", line 4
    esle:
         ^
SyntaxError: invalid syntax
```

In [5]:
```
# Primer: napačen indentation
if flag:
print("Flag is set!")
```

```
  File "<ipython-input-5-2009e1311970>", line 3
    print("Flag is set!")
    ^
IndentationError: expected an indented block
```

Runtime errors

Primer runtime errors:

- Deljenje z 0
- Dostopanje do elementov, ki ne obstajajo
- Dostopanje do datotek, ki ne obstajajo

> - division by zero

- performing an operation on incompatible types
- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn't exist
- trying to access a file which doesn't exist

In [6]:
```
# Primer: deljenje z 0
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-6-eca1cc1fcdbe> in <module>
      1 # Primer: deljenje z 0
----> 2 1 / 0

ZeroDivisionError: division by zero
```

Logical errors

Logične napake nam povzročijo napačne rezultate. Program je lahko sintaksično pravilno zapisan ampak nam ne bo vrnil iskanega rezultata.

Primeri

- Uporabna napačne spremenljivke
- napačna indentacija
- uporaba celoštevilskega deljenja in ne navadnega deljenja

> Logical errors are the most difficult to fix. They occur when the program runs without crashing, but produces an incorrect result. The error is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. You will have to find the problem on your own by reviewing all the relevant parts of your code – although some tools can flag suspicious code which looks like it could cause unexpected behaviour.

In [ ]:

The try and except statements

Da obvladujemo morebitne napake uporabljamo try-except:

In [7]:
```
for _ in range(3):
    x = int(input("Vnesi prvo številko: "))
    y = int(input("Vnesi drugo številko: "))
    rezultat = x / y
    print(f"{x}/{y} = {rezultat}")
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: 2
1/2 = 0.5

Vnesi prvo številko: a

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-b27c485d0cd1> in <module>
      1 for _ in range(3):
----> 2     x = int(input("Vnesi prvo številko: "))
      3     y = int(input("Vnesi drugo številko: "))
      4     rezultat = x / y
      5     print(f"{x}/{y} = {rezultat}")

ValueError: invalid literal for int() with base 10: 'a'
```

In [1]:
```
for _ in range(3):
    x = int(input("Vnesi prvo številko: "))
    y = int(input("Vnesi drugo številko: "))
    rezultat = x / y
    print(f"{x}/{y} = {rezultat}")
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: 2
1/2 = 0.5

Vnesi prvo številko: 1
Vnesi drugo številko: 0


---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [1], in <cell line: 1>()
      2 x = int(input("Vnesi prvo številko: "))
      3 y = int(input("Vnesi drugo številko: "))
----> 4 rezultat = x / y
      5 print(f"{x}/{y} = {rezultat}")
      6 print()

ZeroDivisionError: division by zero
```

Ko se zgodi napaka, Python preveri ali se naša koda nahaja znotraj **try** bloka. Če se ne nahaja, potem bomo dobili error in izvajanje programa se bo ustavilo.

Če se nahaja znotraj try-except blocka, se bo izvedla koda znotraj **except** bloka in program bo nadaljeval z izvajanjem.

In [8]:
```
for _ in range(3):
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except:
        print("Prislo je do napake!")
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: 0
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: a
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: 2
1/2 = 0.5
```

In [ ]:
```

```

Če se je napaka zgodila znotraj funkcije in znotraj funkcije ni bila ujeta (ni bila znotraj try-except bloka), potem gre Python preverjati ali se klic te funkcije nahaja znotraj try-except bloka.

In [9]:
```
def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except:
        print("Prislo je do napake!")

for _ in range(3):
    delilnik()
    print()
```

```
Vnesi prvo številko: a
Prislo je do napake!

Vnesi prvo številko: a
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: 2
1/2 = 0.5
```

In [10]:
```
def delilnik():
    x = int(input("Vnesi prvo številko: "))
    y = int(input("Vnesi drugo številko: "))
    rezultat = x / y
    print(f"{x}/{y} = {rezultat}")

for _ in range(3):
    try:
        delilnik()
    except:
        print("Prislo je do napake!")
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko:
Prislo je do napake!

Vnesi prvo številko: s
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: 0
Prislo je do napake!
```

In [ ]:

In [ ]:

Naloga:

Napišite funkcion **fakulteta**, ki uporabnika vpraša naj vnese cifro in izračuna fakulteto te cifre. Fakulteta se izračuna: 3! = 3*2*1 = 6

Funkcija naj vrne rezultat. Oziroma, če uporabik ni vnesel številke naj funkcija ponovno zahteva od uporabnika vnos cifre.

```
INPUT:
print(fakulteta())

OUTPUT:
Vnesi cifro: a
To ni bila številka.
Vnesi cifro: b
To ni bila številka.
Vnesi cifro: 3
6
```

In [5]:
```
def fakulteta():
    while True:
        try:
            num = int(input("Vnesi cifro: "))
            rezultat = 1
            for i in range(1, num+1):
                #print(i)
                rezultat *= i
            return rezultat
        except:
            print("To ni bila številka.")

print(fakulteta())
```

```
Vnesi cifro: 5
120
```

In [ ]:
```

```

Handling an error as an object

In [12]:
```
def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except:
        print("Prislo je do napake!")

for _ in range(3):
    delilnik()
    print()
```

```
Vnesi prvo številko: a
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: 0
Prislo je do napake!

Vnesi prvo številko: 1
Vnesi drugo številko: 2
1/2 = 0.5
```

Tako kot sedaj hendlamo error ne dobimo nobenega podatka o errorju nazaj. Ne vemo zakaj je prišlo do napake in do kakšne napake je prišlo.

In [13]:
```
def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except Exception as e:
        print("Prislo je do napake!")
        print(type(e))
        print(e)

for _ in range(3):
    delilnik()
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: a
Prislo je do napake!
<class 'ValueError'>
invalid literal for int() with base 10: 'a'

Vnesi prvo številko: 1
Vnesi drugo številko: 0
Prislo je do napake!
<class 'ZeroDivisionError'>
division by zero

Vnesi prvo številko: 2
Vnesi drugo številko: 1
2/1 = 2.0
```

Handling different errors differently

In [14]:
```python
def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except Exception as e:
        print("Prislo je do napake!")
        print(type(e))
        print(e)

for _ in range(3):
    delilnik()
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: a
Prislo je do napake!
<class 'ValueError'>
invalid literal for int() with base 10: 'a'

Vnesi prvo številko: 1
Vnesi drugo številko: 0
Prislo je do napake!
<class 'ZeroDivisionError'>
division by zero

Vnesi prvo številko: 2
Vnesi drugo številko: 3
2/3 = 0.6666666666666666
```

V našem primeru sedaj hendlamo katerikoli **Exception** na enak način.

Lahko pa različne errorje hendlamo na različni način.

Preprosto dodamo še en except stavek.

In [15]:
```
def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except ValueError as e:
        print("Obe spremenljivki morata biti številki!")
        print(type(e))
        print(e)
    except ZeroDivisionError as e:
        print("Druga številka ne sme biti 0!")
        print(type(e))
        print(e)

for _ in range(3):
    delilnik()
    print()
```

```
Vnesi prvo številko: 1
Vnesi drugo številko: a
Obe spremenljivki morata biti številki!
<class 'ValueError'>
invalid literal for int() with base 10: 'a'

Vnesi prvo številko: 1
Vnesi drugo številko: 0
Druga številka ne sme biti 0!
<class 'ZeroDivisionError'>
division by zero

Vnesi prvo številko: 2
Vnesi drugo številko: 1
2/1 = 2.0
```

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 bo program crashnu.

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

> except clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which it is derived).

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

(Se pokaže kako ma Python zgrajeno hierarhijo dedovanja Errorjev). 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
  - UserWarning
  - FutureWarning
  - ImportWarning
  - UnicodeWarning
  - BytesWarning
  - ResourceWarning

In [16]:
```
import inspect

def delilnik():
    try:
        x = int(input("Vnesi prvo številko: "))
        y = int(input("Vnesi drugo številko: "))
        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except Exception:
        print("Zmeraj ta prestreže.")
    except ValueError:
        print("Obe spremeljivki morata biti številki.")
    except ZeroDivisionError:
        print("Deljitelj ne sme biti 0.")

for _ in range(3):
    delilnik()
    print()


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

```
Vnesi prvo številko: 1
Vnesi drugo številko: 0
Zmeraj ta prestreže.

Vnesi prvo številko: 1
Vnesi drugo številko: a
Zmeraj ta prestreže.

Vnesi prvo številko: a
Zmeraj ta prestreže.

(<class 'Exception'>, <class 'BaseException'>, <class 'object'>)
(<class 'ValueError'>, <class 'Exception'>, <class 'BaseException'>, <class 'ob
ject'>)
(<class 'ZeroDivisionError'>, <class 'ArithmeticError'>, <class 'Exception'>, <
class 'BaseException'>, <class 'object'>)
```