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]:

```
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)](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 'object'>)
(<class 'ZeroDivisionError'>, <class 'ArithmeticError'>, <class 'Excep
tion'>, <class 'BaseException'>, <class 'object'>)
```

In [ ]:

```

```

Raising exceptions

Napake lahko rais-amo tudi sami.

In [18]:

```
def delilnik_pozitivnih_st():
    try:
        x = int(input("Vnesi prvo številko: "))
        if x < 0:
            raise ValueError("Vnešena mora biti pozitivna številka")

        y = int(input("Vnesi drugo številko: "))
        if y < 0:
            raise ValueError("vnešena mora biti pozitivna številka")

        rezultat = x / y
        print(f"{x}/{y} = {rezultat}")
    except ValueError as e:
        print(e)
    except ZeroDivisionError:
        print("Deljitelj ne sme biti 0.")

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

```
Vnesi prvo številko: -1
Vnešena mora biti pozitivna številka

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

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

V tem primeru lahko uporabnik vnese negativno številko in ne bomo dobili errora pri pretvorbi:

```
int(input("Vnesi število: "))
```

Zato smo sami dodali preverjanje ali je številka pozitivna ali ne. V primeru, ko številka ni pozitivna smo sami vzdignili **ValueError** z našim specifičnim sporočilom.

In [ ]:

Naloga:

Napišite funkcijo **is_palindrom**, ki od uporabnika zahteva naj vnese besedo. Funkcija naj vrne True, če je beseda palindrom, v nasprotnem primeru False. Palindrom je beseda, ki se prebere isto od leve proti desni in od desne proti levi.

Če uporabnik vnese samo številke naj funkcija rais-a ValueError.

Program naj 3x zažene funkcijo. V kolikor pride do ValueError naj se izpiše sporočilo in izvajanje programa nadaljuje.

```
OUTPUT:
Vnesi besedo: Ananas
The word is NOT palindrom.

Vnesi besedo: 1234
Vnešene so bile samo številke.

Vnesi besedo: racecar
The word is PALINDROM
```

In [ ]:

```
def is_palindrom():
    beseda = input("Vnesi besedo: ")
    if beseda.isnumeric():
        raise ValueError("Vnešene so bile samo številke.")

    st_crk = len(beseda)
    for i in range(st_crk):
        #print("Checking", beseda[i], "and", beseda[-1*i-1])
        if not(beseda[i] == beseda[-1*i -1]):
            return False
    return True

for _ in range(3):
    try:
        if is_palindrom():
            print("The word is PALINDROM")
        else:
            print("The word is NOT palindrom.")
    except ValueError as e:
        print(e)
    print()
```

In [ ]:

```

```

The else and finally statements

Skupaj z try-except lahko uporabimo tudi `else` in `finally` .

`else` se bo izvršil, če try ne vrže napake.

In [19]:

```
try:
    x = int(input("Vnesi številko: "))
except ValueError:
    print("To ni številka.")
else:
    print("Else statement.")

print("End")
```

```
Vnesi številko: 1
Else statement.
End
```

`finally` se izvede po koncu try-except ne glede ali se je napaka ni zgodila, ali se je napaka zgodila in je bila pohendlana, ali se je napaka zgodila in ni bila pohendlana.

Ponavadi se uporabi za čiščenje kode.

In [20]:

```
try:
    x = int(input("Vnesi številko: "))
    print(5/x) # da simuliramo deljenje z 0, ki bo naš nepohendlan error
except ValueError:
    print("To ni številka.")
finally:
    print("Finally statement.")

print("End")
```

```
Vnesi številko:
To ni številka.
Finally statement.
End
```

Writting our own Exceptions

Napišemo lahko tudi naše Exceptions.

https://www.programiz.com/python-programming/user-defined-exception (https://www.programiz.com/python-programming/user-defined-exception)

Svoje exceptione lahko ustvarimo tako, da ustvarimo nov razred, ki deduje iz nekega Exception razreda. Ponavadi je to kar direktno iz osnovnega **Exception** razreda.

In [21]:

```
class MojError(Exception):
    pass

try:
    raise MojError("We raised MojError")
except MojError as e:
    print(e)
```

```
We raised MojError
```

Ko pišemo bolj obsežen python program, je dobra praksa, da vse naše errorje zapišemo v posebno datoteko. Ponavadi je datoteka poimenovana **errors.py** ali **exceptions.py**.

In [ ]:

```

```

Če si pogledamo na bolj konkretnem primeru:

Ustvarili bomo program, kjer uporabnik ugiba neko določeno celo številko. Ustvarili bomo dva naša error classa. Enega v primeru, če je ugibana številka prevelika, drugega v primeru, da je ugibana številka premajhna.

In [23]:

```
class VrednostPremajhna(Exception):
    pass


class VrednostPrevisoka(Exception):
    pass


number = 10 # številka katero ugibamo

while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise VrednostPremajhna
        elif i_num > number:
            raise VrednostPrevisoka
        break
    except VrednostPremajhna:
        print("Ugibana vrednost je premajhna!")
        print()
    except VrednostPrevisoka:
        print("Ugibana vrednost je previsoka!")
        print()

print("PRAVILNO.")
```

```
Enter a number: 20
Ugibana vrednost je previsoka!

Enter a number: 1
Ugibana vrednost je premajhna!

Enter a number: 5
Ugibana vrednost je premajhna!

Enter a number: 11
Ugibana vrednost je previsoka!

Enter a number: 9
Ugibana vrednost je premajhna!

Enter a number: 10
PRAVILNO.
```

In [ ]:

```

```

Naloga:

Napišite funkcijo, ki kot parameter **x** premje neko celo število. Funkcija naj izpiše zadnjih x vrstic v datoteki *naloga2.txt*.

```
INPUT:
funkcija(3)

OUTPUT:
line 7
line 8
line 9
```

def funkcija(n): with open("naloga2.txt", "r") as f: data = f.readlines() for line in data[-n:]: print(line, end="")

funkcija(3)

Naloga:

Napišite funkcijo, ki v datoteko *naloga5.txt* zapiše vse datume, ki so **petek 13.** v letih od 2020 do 2030.

Da najdete datume si lahko pomagate s knjižnjico datetime.

```
INPUT:
print(fakulteta())

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

from datetime import date

def funkcija(): with open("naloga5.txt", "w") as f: for year in range(2020, 2030): for month in range(1, 13): datum = date(year, month, 13) #print(datum) if datum.weekday() == 4: # 0=Mon, 1=Tue, ..., 4=Fri f.write(f'{datum.strftime("%d. %b %Y")}\n')

funkcija()

In [ ]:

In [ ]: