Današnja naloga bo neka preprosta analiza potnikov na Titaniku.

Podatke bomo uvozili v naš program, jih grafično prikazali in opravili kratko analizo.

Delo z datotekami

Datoteke uporabljamo, da v njih trajno shranimo podatke.

V splošnem delo z datotekami poteka na sledeč način:

- Odpremo datoteko
- Izvedemo operacijo (pisanje podatkov v datoteko, branje podatkov, itd..)
- Zapremo datoteko (ter tako sprostimo vire, ki so vezani na upravljanje z datoteko -> spomin, procesorska moč, itd..)

Odpiranje datotek

Python ima vgrajeno funkcijo `open()` za odpiranje datotek.

Funkcija nam vrne `file object` , imenovan tudi **handle** , s katerim lahko izvajamo operacije nad datoteko.

```
f = open("test.txt")    # Odpre datoteko v trenutnem direktoriju
```

Če pa naše datoteke, ki jo želimo odpreti, v trenutnem direktoriju ni, moramo pot do datoteke definirati:

```
# Oblika: "path/to/my/file.txt"
f = open("C:/Users/38664/Desktop/path.txt")  # -> Definiramo pot do datoteke
```

Pri kopiranju poti v Windows-u previdno:

```
# Hitro se zgodi to:
f = open("C:\Users\38664\Desktop\path.txt") # SyntaxError
```

Zato moramo biti previdni in našo pot primerno urediti s tem, da dodamo še en back-slash ob vsak back-slash v naši poti.

> Spomnimo: znak V Pythonu znak '\' (back-slash) nekako napoveduje poseben znak (recimo \n -> newline), torej da našemu nizu (string) dopovemo, da bi radi \ znak in z njim nočemo napovedat posebnega znaka, moramo dodati še en back-slash.

```
f = open("C:\\Users\\38664\\Desktop\\path.txt")
```

Dodatno lahko specificiramo v kakšnem načinu želimo odpreti datoteko.

Lahko jo odpremo v **text mode**. Ko beremo podatke v tem načinu, dobivamo *strings*. To je *default mode*. Lahko pa datoteko odpremo v **binary mode**, kjer podatke beremo kot *bytes*. Takšen način se uporablja pri branju non-text datotek, kot so slike, itd..

Datoteke lahko odpremo v načinu:

- **r** - Podatke lahko samo beremo. (default način)
- **w** - Podatke lahko pišemo v datoteko. Če datoteka ne obstaja jo ustvarimo. Če datoteka obstaja jo prepišemo (če so bli noter podatki jih izgubimo)
- **x** - Ustvarimo datoteko. Če datoteka že obstaja operacija fail-a
- **a** - Odpremo datoteko z namenom dodajanja novih podatkov. Če datoteka ne obstaja jo ustvarimo.
- **t** - odpremo v "text mode" (dafult mode)
- **b** - odpremo v "binary mode"

```
f = open("test.txt")      # Ekvivalent 'r' ali 'rt'
```

```
# Datoteko beremo v tekstovni obliki
f = open("test.txt",'r')
print(type(f))
print(f)
```

```
# V datoteko pišemo v tekstovni obliki
f = open("test.txt",'w')
```

V računalništvu imajo črke in drugi znaki v ozadju določene številčne vrednosti, vendar teh vrednosti sami po sebi ne vsebujejo. Na primer, črka **a** ne pomeni številke 97, dokler ni kodirana z določenim kodirnim sistemom (encoding), kot je ASCII ali kakšno drugo. ASCII je kodirni sistem, ki določa številčne vrednosti za znake, vendar obstajajo tudi drugi sistemi kodiranja.

Privzeta kodirni sistem (encoding), ki jo uporablja računalnik, je odvisna od operacijskega sistema. Na primer, Windows uporablja 'cp1252', medtem ko Linux privzeto uporablja 'utf-8'. To pomeni, da lahko program deluje različno na različnih platformah, če se zanese samo na privzeto kodiranje.

Zato je pomembno, da pri odpiranju datotek v besedilnem načinu določimo vrsto kodiranja. Na ta način bo naš program deloval enako na vseh platformah.

```
f = open("test.txt",mode = 'r',encoding = 'utf-8')
```

Zapiranje datotek

Ko končamo z operacijami, moramo datoteko zapreti, saj tako sprostimo vire, ki so vezani na uporabo datoteke (spomin, procesorska moč, itd..):

```
# 1. Datoteko odpremo
f = open("test.txt", "a")
```

```
# 2. Izvedemo operacije z datoteko
# ...

# 3. Datoteko zapremo
f.close()
```

> Na Linuxu je datotečni opisovalec (file descriptor) pogosto še vedno odprt tudi po klicu f.close(), dokler ni dosežen operacijski sistemski limit, kar lahko postane problem pri dolgoročnem delovanju skript.

Tak način upravljanja z datotekami ni najbolj varen. Če smo odprli datoteko in potem med izvajanjem operacije nad datoteko pride do napake, datoteke ne bomo zaprli.

Varnejši način bi bil z uporabo `try-finally`.

```
try:
    # Poskusi odpreti datoteko
    f = open("test.txt", "a")
    # V primeru napake takoj skoči v finally blok in datoteko zapre
finally:
    f.close()
```

```
try:
    f = open("test.txt", "a")
    # ...
    # Sprožimo napako, zaradi katere bo Python takoj skočil v finally blok in zaprl datoteko
    raise ValueError
finally:
    f.close()
```

```
try:
    f = open("text.txt", "a") # Odpremo datoteko
    x = int("neštevilka") # Ta del kode bo povzročil nenamerno napako (torej smo se zmotili pr
    print("Ta vrstica ne bo izvedena!") # Ker je prišlo do napake v eni vrstici prej, Python t
finally:
    f.close() # Zapre datoteko
```

> Zakaj zapirati datoteke? : Python with Context Managers

Isto stvar dosežemo z uporabo `with statement`.

> 'with' statement samodejno poskrbi za zaprtje datoteke, ko so vse operacije opravljene. To pomeni, da ne potrebujemo ročno klicati 'f.close()', kar zmanjšuje možnost napak.

```
with open("test.txt", "a") as f:
    # Operacije z datoteko
```

```
    pass
```

V spodnji kodi torej odpiramo datoteko "test.txt" v načinu "a" in jo označimo kot "f"

> Ponovitev: Odpiranje v načinu "a" (append) omogoča, da dodajamo nove vrstice brez prepisovanja obstoječih podatkov

```
with open("test.txt", "a") as f:
    # Tukaj lahko izvedemo operacije z datoteko
    print(f"Je datoteka odprta? {not f.closed}")  # Preverimo, ali je datoteka odprta med oper

# Po zaključku bloka je datoteka zaprta
print(f"Je datoteka zaprta? {f.closed}")
```

Če naredimo napako, nam `with` datoteko še vedno zapre. Spet preverimo s flagom.

```
with open("test.txt", "a") as f:
    f.write("Nekaj besedila\n")
    # Tukaj povzročimo napako (npr. deljenje z nič)
    result = 10 / 0  # Napaka (ZeroDivisionError)

# Preverimo, če se je datoteka zaprla
print(f"Je datoteka zaprta po napaki? {f.closed}")
```

Branje datotek

Za branje, datoteko odpremo v *read* (**r**) načinu.

```
with open("test.txt",'r') as f:
    file_data = f.read()
    print(file_data)
```

```
Hello World!
This is my file.
```

```
with open("test.txt", "r") as f:
    file_data = f.read(2)
    print(file_data)
    file_data = f.read(6)
    print(file_data)
    file_data = f.read()
    print(file_data)
    file_data = f.read()
    print(file_data)
```

```
He
llo Wo
rld!
This is my file.
```

> V programiranju boste velikrat vidl kratico EOF, ki pomeni end of file torej konec datoteke

KAZALEC

Ko v Pythonu delamo z datotekami, se moramo nekako premikati po naši odprti datoteki. Za to imamo tudi ime in sicer **kazalec** (ang. cursor).

Ta označuje naš trenutni položaj v datoteki. Ko datoteko odpremo, se kazalec po defaultu nastavi na začetek (na ničtem mesto). Ko beremo ali pišemo, se kazalec premika naprej.

Po datoteki se lahko tudi premikamo z uporabo `seek()` in `tell()` metode. Z metodo 'seek()' se premaknemo na določeno mesto, s 'tell()' pa dobimo informacije o trenutni poziciji kazalca.

```
with open("test.txt", "r") as f:
    print(f.tell()) # Pove nam trenutno pozicijo kazalca
    f.read(4) # Prebere 4 znake
    print(f.tell())
```

```
0
4
```

```
with open("test.txt", "r") as f:
    print(f.tell())

    reading = f.read(6)
    print(reading)
    print(f.tell())

    f.seek(0)
    print(f.tell())

    reading = f.read(6)
    print(reading)
```

```
0
Hello
6
0
Hello
```

Datoteko lahko hitro in učinkovito preberemo vrstico po vrstico, z uporabo `for` zanke.

```
with open("test.txt", "r") as f:
    for line in f:
        print(line) # Vrstice v datoteki imajo newline znak '\n'
```

```
Hello World!

This is my file.
```

Alternativno lahko uporabljamo `readline()` metodo za branje individualnih vrstic.

Metoda prebere podatke iz datoteke do `newline (\n)` znaka.

```
with open("test.txt", "r") as f:
    print(f.readline())
    print(f.readline())
    print(f.readline())
```

```
Hello World!

This is my file.
```

`readlines()` nam vrne list preostalih linij v datoteki.

Pozor! To nam sprinta list, kjer je vsak elemennt v listu vrstica v naši datoteki. S tem nam Python kazalec postavi na konec.

```
with open("test.txt","r") as f:
    list_of_lines = f.readlines()
    print(list_of_lines)
    print(list_of_lines[1])
```

```
with open("test.txt") as f:
    for line in f.readlines():
        print(line)
```

Naloga:

Napišite funkcijo, ki kot parameter **x** prejme 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 **dictionary**, ki vpraša uporabnika, naj vnese določen string in nato vrne vse besede, ki vsebujejo podani string.
Vse možne besede najdete v datoteki *words_alpha.txt*

```
INPUT:
dictionary()

OUTPUT:
Vnesi besedo: meow
homeown
homeowner
homeowners
meow
meowed
meowing
meows
```

```
def dictionary():
    # 2. Uporabnika prosimo za vnos
    beseda = input("Vnesi besedo: ")

    # 3. Odpremo datoteko
    with open("words_alpha.txt", "r") as f:
        for line in f.readlines():
            if beseda in line:
                print(line, end="")

dictionary()
```

```
homeown
homeowner
homeowners
meow
meowed
meowing
meows
```

Pisanje datotek

Za pisanje v datoteko jo odpremo v načinu za pisanje:

- **w** (ta način bo prepisal vse podatke že shranjene v datoteki)
- **a** (s tem načinom bomo dodajali podatke na konec datoteke)
- **x** (s tem ustvarimo datoteko in lahko začnemo v njo pisati)

> Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

Paziti moramo, da na koncu vsake vrstice ročno dodamo newline znak (\n), da ločimo vrstice.

```
with open("test.txt",'w') as f:
    f.write("my first file\n")
    f.write("This file\n\n")
    f.write("contains three lines\n")
```

```
with open("test.txt",'a') as f:
    f.write("We are adding another line.")
    x = f.write("And another one")
```

Število zapisanih znakov lahko preverimo na naslednji način:

```
with open("test.txt",'a') as f:
    f.write("We are adding another line.")
    x = f.write("And another one")
    print(x)
```

```
15
```

Poskusimo še način 'x'. Z načinom 'x' ustvarimo datoteko. V primeru, da datoteka z enakim imenom že obstaja, nam vrže error.

```
with open("test2.txt", "x") as f:
    f.write("New .txt")
```

Importing

`Importing` je način, kako lahko kodo iz ene datoteke / modula / paketa uporabimo v drugi datoteki / modulu.

- **module** je datoteka, ki ima končnico *.py*
- **package** je direktorij, ki vsebuje vsaj en modul

Importing je zelo pomembna funkcionalnost programiranja, saj nam omogoča uvažanje marsikaterih funkcij, da je delo v Pythonu lažje.

Da importiramo modul uporabimo besedo `import` .

```
import moj_modul
```

Python sedaj najprej 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 ime najde, lahko modul sedaj uporabljamo v naši datoteki.

Za začetek bomo importirali **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

```
import math

moja_spremenljivka = 5
print(dir())

print(moja_spremenljivka)
print(math)
```

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

```
import math

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

Pomagamo si lahko tudi z `help(ime_modula)`

S to funkcijo se nam v terminalu izpišejo vse funkcije modula z opisom.

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

```
import math

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

Naloga:

S pomočjo **math** modula izračunajte logaritem 144 z osnovo 12.
https://docs.python.org/3/library/math.html

```
import math

math.log(144, 12)
```

```
2.0
```

Importing our own module

Module lahko ustvarimo tudi sami.

Začeli bomo s preprosto datoteko, ki jo bomo poimenovali kar `moj_modul.py`

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

```
├── _python_tecaj/
    ├── moj_modul.py
    └── skripta.py
```

V moj_modul.py datoteko bomo definirali razred Pes z njegovim konstruktorjem.

Poleg tega bomo dodali še funkcijo seštevalnik, kjer bomo vrnili vsoto dveh vhodnih parametrov.

Prav tako bomo definirali moja_spremenljivka.

moj_modul.py

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


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


moja_spremenljivka = 100
```

skripta.py

```
import moj_modul

print(dir())
print(dir(moj_modul))

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

print(moj_modul.sestevalnik(5, 6))

print(moj_modul.moja_spremenljivka)
```

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
```

```
    ├── _python_tecaj/
        ├── moj_modul.py
        ├── skripta.py
        └── _moj_package/
            ├── modul2.py
```

modul2.py

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

spremenljivka2 = 200
```

V `skripta.py` bomo klicali modul `modul2.py` .

skripta.py

```
from moj_package import modul2

print(dir())

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

Namesto da uvozimo celoten modul, se lahko s piko premaknemo globlje v direktoriju in iz modula izberemo specifičen objekt / funkcijo / spremenljivko:

```
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.")
```

```
Rešitev pravilna.
Rešitev pravilna.
Rešitev pravilna.
Rešitev pravilna.
```

```python
# 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 ne vemo kaj vse smo importirali in lahko na tak način ponesreči nekaj spremenimo. V našem primeru smo povozili spremenljivko pi iz modula math.

```python
from math import *

print(dir())
print(pi)

pi = 3 # Spremenili smo vrednost spremenljivke pi
print(pi)
```

Naloga:

Napišite funkcijo, ki v datoteko *naloga_petek13.txt* zapiše vse datume, ki so **petek 13.** v letih od 2020 do (brez) 2030.
Da najdete datume si lahko pomagate s knjižnjico datetime.

```
13. Mar 2020
13. Nov 2020
13. Aug 2021
13. May 2022
13. Jan 2023
13. Oct 2023
13. Sep 2024
13. Dec 2024
13. Jun 2025
13. Feb 2026
13. Mar 2026
13. Nov 2026
13. Aug 2027
13. Oct 2028
```

```
13. Apr 2029
13. Jul 2029
```

```python
from datetime import date

def funkcija():
    with open("naloga_petek13.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()
```

URLs

URL (Uniform Resource Locator) je naslov vira na internetu. Python omogoča delo z URL-ji z naslednjimi knjižnicami:

- **`urllib`**: Standardna knjižnica za delo z URL-ji.
- **`requests`**: Priljubljena knjižnica za delo s spletnimi viri.

Knjižnica `urllib.parse` omogoča analizo in urejanje URL-jev.

Funkcija `urlparse` razdeli URL na komponente:

```python
from urllib.parse import urlparse

url = "https://www.example.com:8080/path/to/page?query=test#fragment"
parsed_url = urlparse(url)

print(parsed_url)
# Izpis: ParseResult(scheme='https', netloc='www.example.com:8080', path='/path/to/page', para
```

```
ParseResult(scheme='https', netloc='www.example.com:8080', path='/path/to/page', params='', q
uery='query=test', fragment='fragment')
```

Funkcija `urlunparse` omogoča sestavo URL-jev iz komponent:

```python
from urllib.parse import urlunparse

komponente = ('https', 'www.example.com', '/path/to/page', '', 'query=test', 'fragment')
sestavljen_url = urlunparse(komponente)

print(sestavljen_url)  # Izpis: https://www.example.com/path/to/page?query=test#fragment
```

Knjižnica `requests` poenostavi prenos podatkov z URL-jev.

```python
from urllib import requests

odgovor = requests.get("https://api.github.com")
print(odgovor.status_code)  # Izpis: 200
print(odgovor.text)  # Telo odgovora kot besedilo
```

Dodajanje parametrov URL-ju je prav tako mogoče:

```python
parametri = {"q": "python", "page": 1}
odgovor = requests.get("https://www.example.com/search", params=parametri)

print(odgovor.url)  # Izpis: https://www.example.com/search?q=python&page=1
```

Knjižnica `requests` omogoča prenos datotek ali slik z URL-jev.

```python
url = "https://www.example.com/sample.txt"
odgovor = requests.get(url)

with open("sample.txt", "wb") as file:
    file.write(odgovor.content)
print("Datoteka prenesena!")
```

Knjižnica `validators` omogoča preverjanje, ali je URL veljaven:

```python
from urllib import validators

url = "https://www.example.com"
if validators.url(url):
    print("Veljaven URL!")
else:
    print("Neveljaven URL!")
```

> Namestitev knjižnice `validators`: `pip install validators`

JSON

Malo o delu z JSON v Python: https://realpython.com/python-json/

JSON - JavaScript Object Notation, je način zapisa informacij v organizirano in preprosto strukturo, ki je lahko berljiva tako za ljudi kot tudi za računalnike.

```
{
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ["running", "sky diving", "singing"],
    "age": 35,
    "children": [
```

```
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

Za manipuliranje z JSON podatki v Pythonu uporabljamo `import json` modul.

Python object translated into JSON objects

Python	JSON
`dict`	`object`
`list`, `tuple`	`array`
`str`	`string`
`int`, `long`, `float`	`number`
`True`	`true`
`False`	`false`
`None`	`null`

Primer shranjevanja JSON podatkov.

```
import json
```

```
data = {
    "firstName": "Jane",
    "lastName": "Doe",
    "hobbies": ("running", "sky diving", "singing"),
    "age": 35,
    "children": [
        {
            "firstName": "Alice",
            "age": 6
        },
        {
            "firstName": "Bob",
            "age": 8
        }
    ]
}
```

Da naše podatke spremenimo v JSON zapis, uporabimo metodo `json.dumps(data)`.

Nazaj dobimo string katerega bi lahko zapisali v JSON datoteko.

```
json_data = json.dumps(data)
json_data
```

```
'{"firstName": "Jane", "lastName": "Doe", "hobbies": ["running", "sky diving", "singing"], "a
ge": 35, "children": [{"firstName": "Alice", "age": 6}, {"firstName": "Bob", "age": 8}]}'
```

Da JSON string pretvorimo nazaj v python datatipe uporabimo funkcijo `json.loads(json_string)`.

```
py_data = json.loads(json_data)

print(py_data)
py_data["firstName"]
```

```
{'firstName': 'Jane', 'lastName': 'Doe', 'hobbies': ['running', 'sky diving', 'singing'], 'ag
e': 35, 'children': [{'firstName': 'Alice', 'age': 6}, {'firstName': 'Bob', 'age': 8}]}
'Jane'
```

Če želimo podatke direktno shraniti v `.json` datoteko imamo za to metodo `json.dump()`.

Bodimo pozorni, da funkcija `dump` vzame dva argumenta:

1. Podatke, ki jih želimo zapisati v datoteko
2. Ime datoteke, kamor bodo napisani bajti

```
with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

Da podatke preberemo nazaj iz datoteke, uporabimo metodo `json.load()`.

Potrebno se je zavedati, da konverzija datatipov ni nujno točna.

Python-JSON conversion table

JSON	Python
`object`	`dict`
`array`	`list`
`string`	`str`
`number` (int)	`int`
`number` (real)	`float`
`true`	`True`
`false`	`False`
`null`	`None`

```
with open("data_file.json", "r") as read_file:
    data = json.load(read_file)
    print(data)
    print(type(data))
```

```
{'firstName': 'Jane', 'lastName': 'Doe', 'hobbies': ['running', 'sky diving', 'singing'], 'age': 35, 'children': [{'firstName': 'Alice', 'age': 6}, {'firstName': 'Bob', 'age': 8}]}
<class 'dict'>
```

Naloga:

Napišite program, ki prebere **podatki.json**. Program naj primerja zaslužke vseh oseb med seboj (salary + bonus) in nato izpiše ime in celotni zaslužek te osebe.

```
OUTPUT:
Oseba, ki zasluži največ je martha. Zasluži 10300€.
```

```
import json

with open("podatki.json") as f:
    data = json.load(f)
    #print(data)

max_pay = 0
name = ""
for employee in data["company"]["employees"]:
    print(employee)
    if employee["payble"]["salary"] + employee["payble"]["bonus"] > max_pay:
        name = employee["name"]
        max_pay = employee["payble"]["salary"] + employee["payble"]["bonus"]

print(f"Oseba, ki zasluži največ je {name}. Zasluži {max_pay}€.")
```

```
{'name': 'emma', 'payble': {'salary': 7000, 'bonus': 800}}
{'name': 'derek', 'payble': {'salary': 4000, 'bonus': 1000}}
{'name': 'alex', 'payble': {'salary': 7500, 'bonus': 500}}
{'name': 'susan', 'payble': {'salary': 6300, 'bonus': 350}}
{'name': 'martha', 'payble': {'salary': 9100, 'bonus': 1200}}
{'name': 'clark', 'payble': {'salary': 7700, 'bonus': 270}}
{'name': 'luise', 'payble': {'salary': 8200, 'bonus': 900}}
Oseba, ki zasluži največ je martha. Zasluži 10300€.
```

Titanic Analysis

Za našo nalogo si bomo sedaj pogledali podatke o potnikih ladje Titanic - titanic.csv.

Pogledali si bomo, koliko preživelih potnikov je bilo moških in koliko žensk

Podatki so shranjeni v sledeče:

- **PassengerId** - ID vsakega potnika. Da lahko ločimo potnike med seboj
- **Survived** - 0, če oseba ni preživela. 1, če je oseba preživela.
- **Pclass** - v katerem razredu je bila oseba. 1 - prvi razred, 2 - drugi razred, 3 - tretji razred
- **Name** - ime osebe
- **Last name** - priimek osebe
- **Sex** - spol osebe
- **Age** - starost osebe v letih
- **SibSp** - število sorojencev osebe oziroma partnerjev na ladji
- **Parch** - število staršev oziroma otrok osebe na ladji
- **Ticket** - številka karte osebe
- **Fare** - cena karte
- **Cabin** - številka sobe
- **Embarked** - kje se je oseba vkrcala, C = Cherbourg, Q = Queenstown, S = Southampton

Naloga:

Uvozite podatke. Vsako vrstico posebaj razdelite glede na vejico in jo shranite v `data` list.

Prvih 5 vrstic v listu `data`:

```
['1', '0', '3', '"Braund', ' Mr. Owen Harris"', 'male', '22', '1', '0', 'A/5 21171', '7.25', '', 'S']
['2', '1', '1', '"Cumings', ' Mrs. John Bradley (Florence Briggs Thayer)"', 'female', '38', '1', '0', 'PC 17599', '71.2833', 'C85', 'C']
['3', '1', '3', '"Heikkinen', ' Miss. Laina"', 'female', '26', '0', '0', 'STON/O2. 3101282', '7.925', '', 'S']
['4', '1', '1', '"Futrelle', ' Mrs. Jacques Heath (Lily May Peel)"', 'female', '35', '1', '0', '113803', '53.1', 'C123', 'S']
['5', '0', '3', '"Allen', ' Mr. William Henry"', 'male', '35', '0', '0', '373450', '8.05', '', 'S']
```

```python
data = []

with open("./titanic.csv", "r") as f:
    for line in f.readlines()[1:]: # preskočimo prvo vrstico, ker so to imena stolpcev
        line = line.strip() # odstranimo nepotrebne \n in presledke
        line_splitted = line.split(",")
        data.append(line_splitted)


for line in data[:5]:
    print(line)
```

```
['1', '0', '3', '"Braund', ' Mr. Owen Harris"', 'male', '22', '1', '0', 'A/5 21171', '7.25',
'', 'S']
['2', '1', '1', '"Cumings', ' Mrs. John Bradley (Florence Briggs Thayer)"', 'female', '38',
'1', '0', 'PC 17599', '71.2833', 'C85', 'C']
['3', '1', '3', '"Heikkinen', ' Miss. Laina"', 'female', '26', '0', '0', 'STON/O2. 3101282',
'7.925', '', 'S']
['4', '1', '1', '"Futrelle', ' Mrs. Jacques Heath (Lily May Peel)"', 'female', '35', '1',
'0', '113803', '53.1', 'C123', 'S']
['5', '0', '3', '"Allen', ' Mr. William Henry"', 'male', '35', '0', '0', '373450', '8.05',
'', 'S']
```

Naloga:

Izračunajte koliko % možnosti so imeli moški za preživetje in koliko ženske.

(Primer, % preživetja za moške je "moški survived" / "vsi moški potniki").

```
{'male': {'survived': 109,
      'died': 468,
      'survived_%': 0.18890814558058924,
      'died_%': 0.8110918544194108},
 'female': {'survived': 233,
      'died': 81,
      'survived_%': 0.7420382165605095,
      'died_%': 0.25796178343949044}}
```

```
survived_dist = {"male": {"survived": 0, "died": 0}, "female": {"survived": 0, "died": 0}}

for line in data:
    if int(line[1]): # person survived
        survived_dist[line[5]]["survived"] += 1
    else:
        survived_dist[line[5]]["died"] += 1

survived_dist
```

```
{'male': {'survived': 109, 'died': 468},
 'female': {'survived': 233, 'died': 81}}
```

```
for s in survived_dist:
    total = survived_dist[s]["survived"] + survived_dist[s]["died"]
    survived_dist[s]["survived_%"] = survived_dist[s]["survived"] / total
    survived_dist[s]["died_%"] = survived_dist[s]["died"] / total

survived_dist
```

```
{'male': {'survived': 109,
  'died': 468,
  'survived_%': 0.18890814558058924,
  'died_%': 0.8110918544194108},
 'female': {'survived': 233,
  'died': 81,
  'survived_%': 0.7420382165605095,
  'died_%': 0.25796178343949044}}
```

Third party libraries

Zgornje knjižnice so vgrajene v Python in so naložene ob inštalaciji Python-a.

Kodo, ki jo je napisal nekdo drug, ponavadi najdemo v obliki **3rd party knjižnjice** katero moramo prvo inštalirati.

3rd party knjižnjice/pakete lahko inštaliramo s pomočjo **pip**, ki je python package manager.

Če imamo pip inštaliran, lahko preverimo tako, da v konzolo vpišemo `pip --version``

```
$pip --version

pip 19.1.1 from C:\Users\Anaconda3\lib\site-packages\pip (python 3.7)
```

> WINDOWS: Download get-pip.py to a folder on your computer. Open a command prompt and navigate to the folder containing get-pip.py. Run the following command: `

Če dobimo napako, moramo pip inštalirati: python get-pip.py `

> Debian(Ubuntu/Kali etc) distribution of linux: `$ sudo apt-get install python-pip`

S pomočjo pip lahko inštaliramo knjižnjice s preprostim ukazom

```
pip install <package_name>

pip install matplotlib
```

Matplotlib

Za grafični prikaz bomo tako inštalirali in importirali knjižnjico `matplitlib`.

Je eden izmed najbolj uporabljenih Python paketov za grafično prikazovanje podatkov.

```
pip install matplotlib
```

```
# Uvoz knjižnice za matplotlib
import matplotlib.pyplot as plt

%matplotlib inline
#%matplotlib notebook
# te dve liniji definirata prikazovanje, če uporabljamo jupyter notebook
# to je syntax-a specifična za jupyter notebook
```

`matplotlib.pyplot` je zbirka funkcij, ki delujejo podobno kot MATLAB. Vsaka pyplot funkcija naredi neko spremembo figuri (prikaz), recimo ustvari figuro, ustvari območje izrisovanja v prikazu, izriše linije v območju izrisovanja, doda oznake, ipd.

Simple plot

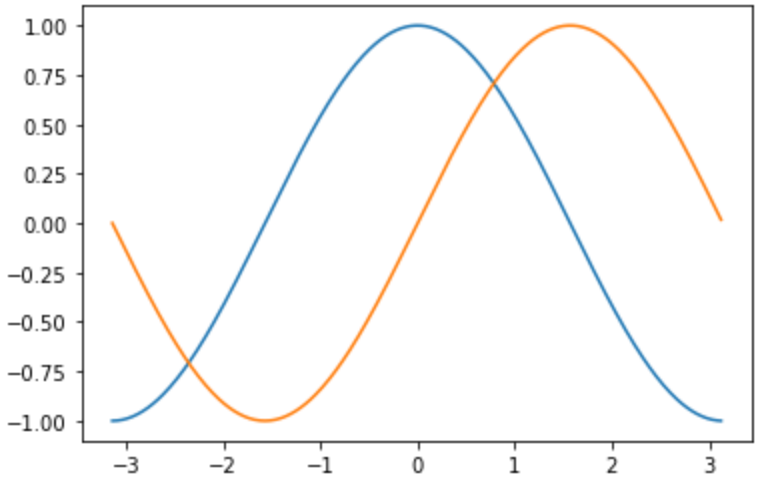
Začeli bomo s preprostim primerom:

> **izrisali bomo funkcijo sinus in kosinus na isti graf**.

Za začetek bomo uporabili raznorazne default vrednosti, za velikost grafa, barvo črt, stil črt, itd.., nato pa bomo graf nadgrajevali.

```
import matplotlib.pyplot as plt
import math
# x-os
x = [math.radians(x) for x in range(-180, 180)]

# y-os, potrebujemo vrednosti cos(x) in sin(x):4.f
cos = [math.cos(i) for i in x]
sin = [math.sin(i) for i in x]

for i, x_ in enumerate(x[::30]):
    print(f"x: {x_:7.4f} \t sin: {sin[i]:7.4f} \t cos: {cos[i]:7.4f}")



plt.plot(x,cos)
plt.plot(x,sin)

plt.show()
```

```
x: -3.1416      sin: -0.0000    cos: -1.0000
x: -2.6180      sin: -0.0175    cos: -0.9998
x: -2.0944      sin: -0.0349    cos: -0.9994
x: -1.5708      sin: -0.0523    cos: -0.9986
x: -1.0472      sin: -0.0698    cos: -0.9976
x: -0.5236      sin: -0.0872    cos: -0.9962
x:  0.0000      sin: -0.1045    cos: -0.9945
x:  0.5236      sin: -0.1219    cos: -0.9925
x:  1.0472      sin: -0.1392    cos: -0.9903
x:  1.5708      sin: -0.1564    cos: -0.9877
x:  2.0944      sin: -0.1736    cos: -0.9848
x:  2.6180      sin: -0.1908    cos: -0.9816
```



`plt.plot(x_podatki, y_podatki)` S tem ukazom na naš plot narišemo podatke.

`plt.show()` pokaže naš graf.

Kako Matplotlib izrisuje stvari (Figures, Subplots, Axes, Ticks)

Matplotplib nam bo po defaultu ustvaril `figure`. Po defaultu bo znotraj `figure` ustvaril 1 `subplot`, ki bo zavzel celotno `figuro` in znotraj `subplota` nam bo izrisal graf.

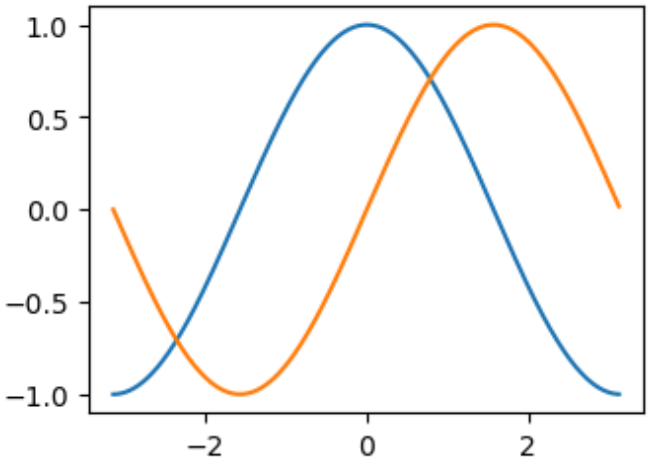
Figure v matplotlib-u pomeni celotno okno. Za naslov imajo `Figure _cifra_`, številčenje pa se začne z 1. Ostali parametri, ki definirajo `figure` so:

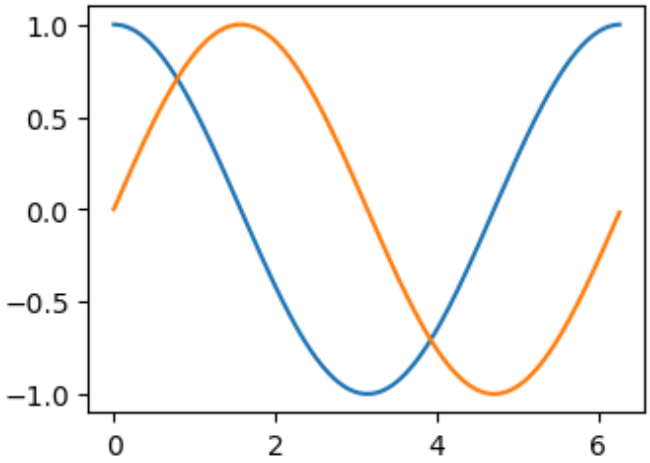
- figsize - velikost figure v inčih (dolžina, višina)
- dpi - resolucija v dots per inch
- facecolor - barva risalne podlage
- edgecolor - barva obrobe

Subplot

Z uporabo `subplot` lahko našo figuro razdelamo na več razdelkov. Specificiramo število vrstic, število stolpcev in številko `plot-a`.


```
import matplotlib.pyplot as plt
import math

plt.figure(figsize=(8,6), dpi=100)

a = plt.subplot(2,2,2)

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]

a.plot(X, C)
a.plot(X, S)

b = plt.subplot(2,2,3)
b.plot(X, C)
b.plot(X, S)

plt.show()
```



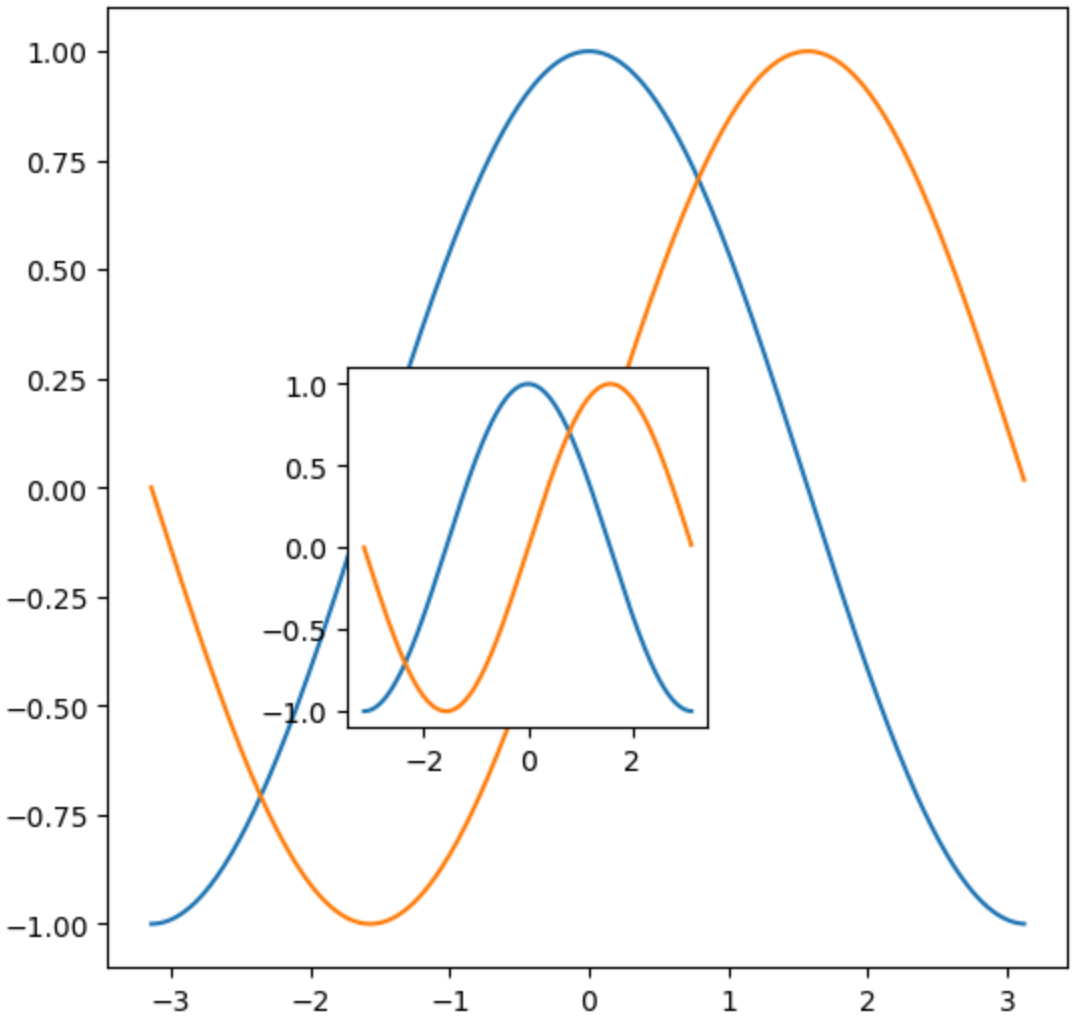


Axes se obnašajo podobno kot `subplot`, le da oni lahko ležijo kjerkoli v figuri.

```
axes([left, bottom, width, height])
```

Width in height sta normalizirana glede na figuro.


```
#import matplotlib.pyplot as plt
#import math

plt.figure(figsize=(6,6), dpi=100)

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]


plt.axes([0,0,0.8, 0.8])
plt.plot(X, C)
plt.plot(X, S)


plt.axes([0.2, 0.2, 0.3,0.3])
plt.plot(X, C)
plt.plot(X, S)

plt.show()
```

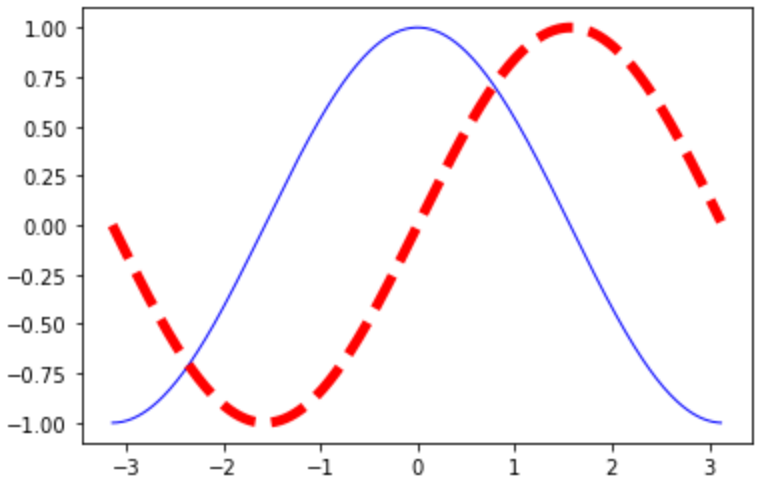


Spreminjanje lastnosti

Spreminjanje lastnosti črt

```
#import matplotlib.pyplot as plt
#import math

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]

# SPREMINJANJE BARVE, DEBELINE IN STIL ČRTE
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
# Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

plt.plot(X, S, color="red", linewidth=5.0, linestyle="--")
#Plotting sinus funkcijo
# color="red" -> barva črte bo rdeča
# linewidth=5.0 -> debelina bo 5.0 pixlov
```

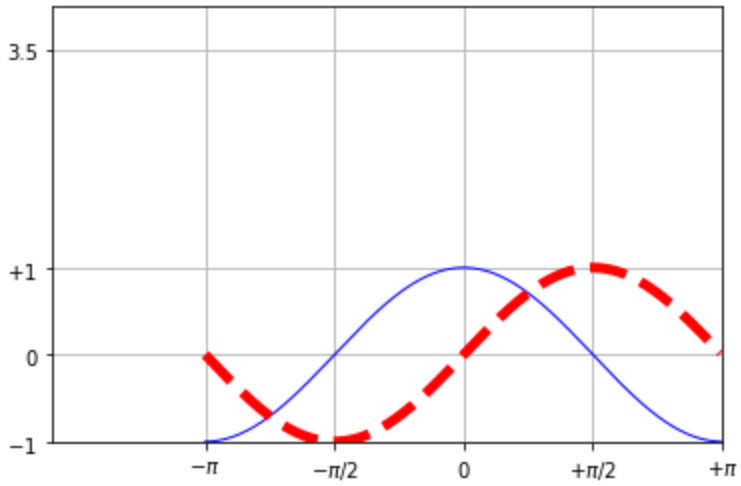
```
# linestyle="--" -> stil črte bo črtkana črta

plt.show()
```



Spreminjanje lastnosti osi

```
#import matplotlib.pyplot as plt
#import math

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]


# SPREMINJANJE BARVE, DEBELINE IN STIL ČRTE
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
#Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

plt.plot(X, S, color="red", linewidth=5.0, linestyle="--")
#Plotting sinus funkcijo
# color="green" -> barva črte bo zelena
# linewidth=5.0 -> debelina bo 5.0 pixlov
# linestyle="--" -> stil črte bo črtkana črta

# OBLIKOVANJE OSI (VELIKOST, VREDNOSTI)
plt.xlim(-5, max(X))
#Postavi meje x osi
# x os bo velika od -5 do max vrednosti X (vključno)

plt.xticks([-3.14, -3.14/2, 0, 3.14/2, 3.14], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'
# Postavi te "oznake" na x osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.ylim(min(C), 4)
# Postavi meje y osi
# y os bo velika od minimalne vrednosti cos do 4 vključno
```

```
plt.yticks([-1, 0, 1, 3.5], [r'$-1$', r'$0$', r'$+1$', '3.5'])
# Postavi te "oznake" na y osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.grid()
# doda pomožne črte na naš graf
# črte izhajajo iz naših x in y

plt.show()
```

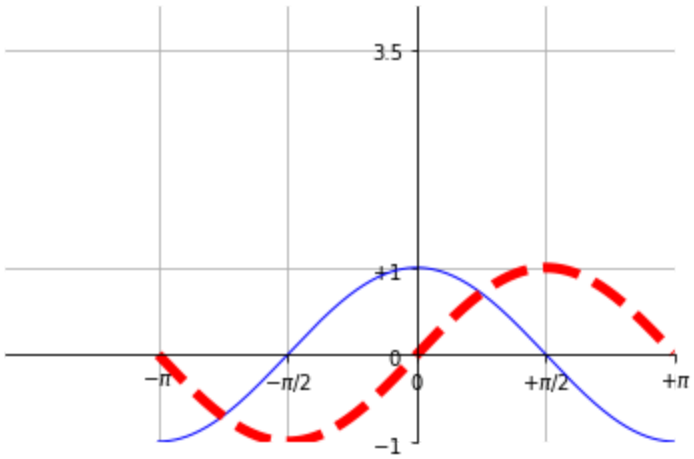


Spreminjanje lastnosti okvira

Črte, ki uokvirjajo naš graf se imenujejo **spines**.

```
#import matplotlib.pyplot as plt
#import math

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]


# SPREMINJANJE BARVE, DEBELINE IN STIL ČRTE
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
#Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

plt.plot(X, S, color="red", linewidth=5.0, linestyle="--")
#Plotting sinus funkcijo
# color="green" -> barva črte bo zelena
# linewidth=5.0 -> debelina bo 5.0 pixlov
# linestyle="--" -> stil črte bo črtkana črta

# OBLIKOVANJE OSI (VELIKOST, VREDNOSTI)
plt.xlim(-5, max(X))
```

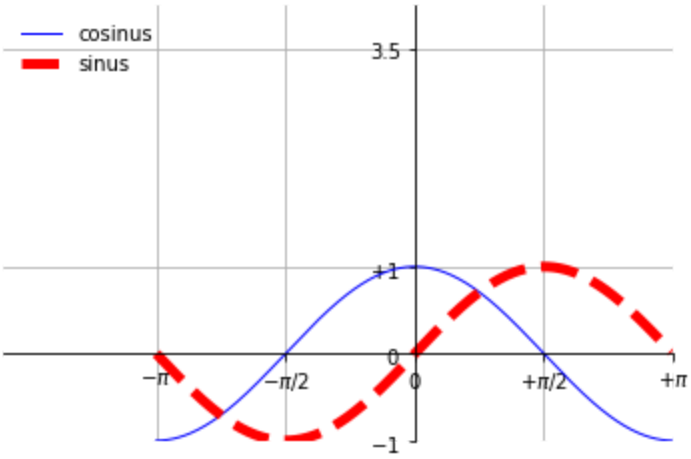
```
#Postavi meje x osi
# x os bo velika od -5 do max vrednosti X (vključno)

plt.xticks([-3.14, -3.14/2, 0, 3.14/2, 3.14], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'
# Postavi te "oznake" na x osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.ylim(min(C), 4)
#Postavi meje y osi
# y os bo velika od minimalne vrednosti cos do 4 vključno

plt.yticks([-1, 0, 1, 3.5], [r'$-1$', r'$0$', r'$+1$', '3.5'])
#Postavi te "oznake" na y osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.grid()
# doda pomožne črte na naš graf
# črte izhajajo iz naših x in y "ticks"

# OBLIKOVANJE OKVIRA
ax = plt.gca() # GetCurrentAxes -> se prav dobimo nš trenutni graf
ax.spines['right'].set_color('none') # našo desno črto napravimo nevidno
ax.spines['top'].set_color('none') # naši zgornjo črto napravimo nevidno
#ax.xaxis.set_ticks_position('top') # naše x oznake lahko premakno iz spodnje na zgornjo črto
ax.spines['bottom'].set_position(('data',0)) # ta nič pomen, da gre skoz 0 na y
#ax.yaxis.set_ticks_position('right') # naše y oznake lahko premaknemo iz leve na desno črto
ax.spines['left'].set_position(('data',0))

plt.show()
```



Dodajanje legende

```
#import matplotlib.pyplot as plt
#import math

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]
```

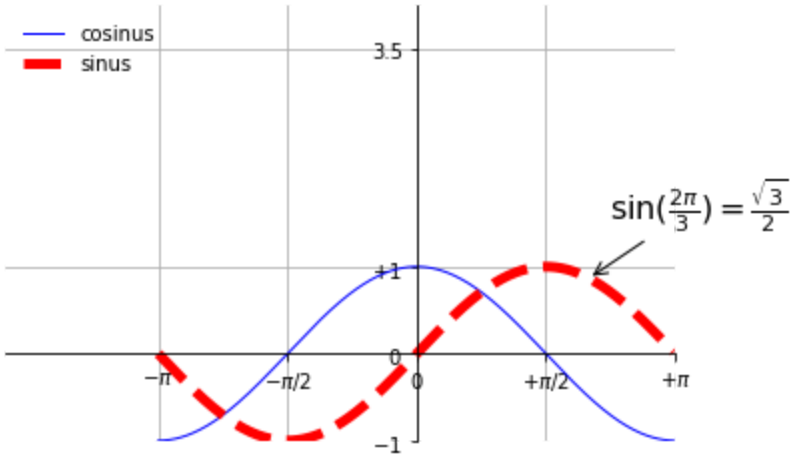
```python
# SPREMINJANJE BARVE, DEBELINE IN STIL ČRTE
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-", label="cosinus")
#Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

plt.plot(X, S, color="red", linewidth=5.0, linestyle="--", label="sinus")
#Plotting sinus funkcijo
# color="green" -> barva črte bo zelena
# linewidth=5.0 -> debelina bo 5.0 pixlov
# linestyle="--" -> stil črte bo črtkana črta

# OBLIKOVANJE OSI (VELIKOST, VREDNOSTI)
plt.xlim(-5, max(X))
#Postavi meje x osi
# x os bo velika od -5 do max vrednosti X (vključno)

plt.xticks([-3.14, -3.14/2, 0, 3.14/2, 3.14], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'
# Postavi te "oznake" na x osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.ylim(min(C), 4)
#Postavi meje y osi
# y os bo velika od minimalne vrednosti cos do 4 vključno

plt.yticks([-1, 0, 1, 3.5], [r'$-1$', r'$0$', r'$+1$', '3.5'])
#Postavi te "oznake" na y osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.grid()
# doda pomožne črte na naš graf
# črte izhajajo iz naših x in y "ticks"

# OBLIKOVANJE OKVIRA
ax = plt.gca() # GetCurrentAxes -> se prav dobimo nš trenutni graf
ax.spines['right'].set_color('none') # našo desno črto napravimo nevidno
ax.spines['top'].set_color('none') # naši zgornjo črto napravimo nevidno
#ax.xaxis.set_ticks_position('top') # naše x oznake lahko premakno iz spodnje na zgornjo črto
ax.spines['bottom'].set_position(('data',0)) # ta nič pomen, da gre skoz 0 na y
#ax.yaxis.set_ticks_position('right') # naše y oznake lahko premaknemo iz leve na desno črto
ax.spines['left'].set_position(('data',0))

# DODAJANJE LEGENDE
plt.legend(loc='upper left', frameon=False)
#doda nam legendo (pred tem mormo našim funkcijam (.plot()) dodat parameter: "label")
# loc -> lokacija
# frameon -> če je uokvirjena legenda ali ne

plt.show()
```



Anotacija

```
#import matplotlib.pyplot as plt
#import math

X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]

# SPREMINJANJE BARVE, DEBELINE IN STIL ČRTE
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-", label="cosinus")
#Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

plt.plot(X, S, color="red", linewidth=5.0, linestyle="--", label="sinus")
#Plotting sinus funkcijo
# color="green" -> barva črte bo zelena
# linewidth=5.0 -> debelina bo 5.0 pixlov
# linestyle="--" -> stil črte bo črtkana črta

# OBLIKOVANJE OSI (VELIKOST, VREDNOSTI)
plt.xlim(-5, max(X))
#Postavi meje x osi
# x os bo velika od -5 do max vrednosti X (vključno)

plt.xticks([-3.14, -3.14/2, 0, 3.14/2, 3.14], [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'
# Postavi te "oznake" na x osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''

plt.ylim(min(C), 4)
#Postavi meje y osi
# y os bo velika od minimalne vrednosti cos do 4 vključno

plt.yticks([-1, 0, 1, 3.5], [r'$-1$', r'$0$', r'$+1$', '3.5'])
#Postavi te "oznake" na y osi
# Prvi argument [] pove katere oznake naj postavi
# Drugi argument [] pove kako naj jih izpiše. V našem primeru izpišemo v formatu Latex r''
```

```
plt.grid()
# doda pomožne črte na naš graf
# črte izhajajo iz naših x in y "ticks"

# OBLIKOVANJE OKVIRA
ax = plt.gca() # GetCurrentAxes -> se prav dobimo ns trenutni graf
ax.spines['right'].set_color('none') # našo desno črto napravimo nevidno
ax.spines['top'].set_color('none') # naši zgornjo črto napravimo nevidno
#ax.xaxis.set_ticks_position('top') # naše x oznake lahko premakno iz spodnje na zgornjo črto
ax.spines['bottom'].set_position(('data',0)) # ta nič pomen, da gre skoz 0 na y
#ax.yaxis.set_ticks_position('right') # naše y oznake lahko premaknemo iz leve na desno črto
ax.spines['left'].set_position(('data',0))

# DODAJANJE LEGENDE
plt.legend(loc='upper left', frameon=False)
#doda nam legendo (pred tem mormo našim funkcijam (.plot()) dodat parameter: "label")
# loc -> lokacija
# frameon -> če je uokvirjena legenda ali ne

# ANOTACIJA -> OPOMBA
t = 2*3.14/3 # vrednost naše točke, na katero hočemo pokazati
plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, math.sin(t)),
             xycoords='data',
             xytext=(10, +30),
             textcoords='offset points',
             fontsize=16,
             arrowprops=dict(arrowstyle="->"))
#r'' -> izpis v latex formatu
#xy=(t, math.sin(t)) -> specificira kero pozicijo anotiramo (točka na kero nej pokaže)
#xycoords="data" -> koordinatni sistem v katerem sta xy=
#xytext -> specificiram kam naj vstavi text
#textcoords -> koordinatni sistem za text -> 'offset points'    offset (in points) from the xy
#arrowprops
#   arrowstyle -> kakšnega stila je naša puščica

plt.show()
```



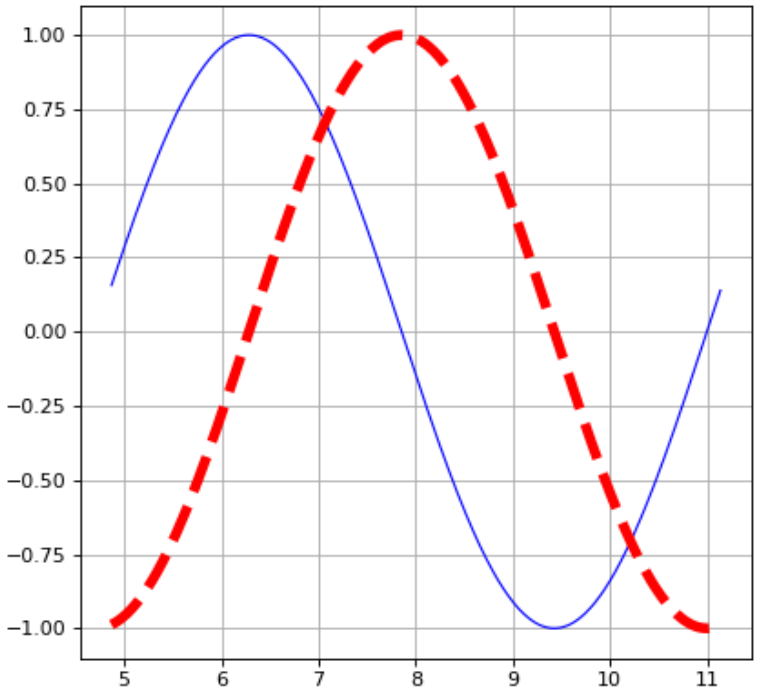
Animation

`FuncAnimation(fig, update, interval=10)`

- fig - figura katero animiramo
- update - funkcija katero se kliče vsak nov frame. Frame spremenljivka je vedno podana kot prva, ostalo so naši poljubni parametri
- interval - delay med frami v milisekindah. Default je 200

--- Teacher's Notes --- Be sure to explain this section thoroughly to ensure students understand the concept.

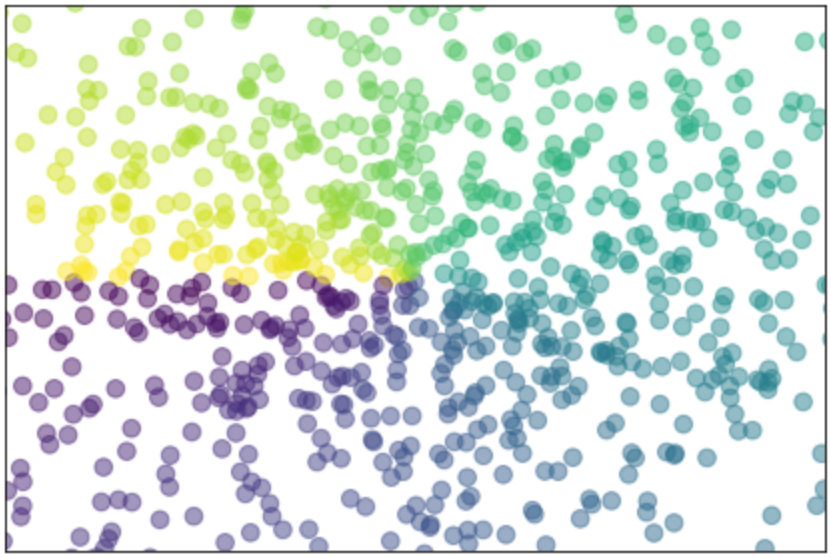
```
#import math
#import matplotlib
#import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
%matplotlib notebook
# zgornja vrstica je potrebna, če delamo na jupyter notebook.
# - če je "inline" se stvari prikazujejo kot statične slike
# - če je "notebook" bo normalno prikazoval animacijo

fig = plt.figure(figsize=(6,6), facecolor='white')

ax = plt.subplot(1,1,1)

X = [math.radians(x) for x in range(-180, 180)]
X_korak = X[-1] - X[-2]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-", label="cosine")
plt.plot(X, S, color="red", linewidth=5.0, linestyle="--",label="sine")


def update(frame):
    global X, X_korak, ax, C, S
    del X[0]
    X.append(X[-1] + X_korak)
    C = [math.cos(i) for i in X]
    S = [math.sin(i) for i in X]
    ax.clear()
    ax.plot(X, C, color="blue", linewidth=1.0, linestyle="-", label="cosine")
    ax.plot(X, S, color="red", linewidth=5.0, linestyle="--",label="sine")
    ax.grid()

animation = FuncAnimation(fig, update, interval = 40) # 40 milisekund je približno 24FPS
plt.show()
```



PRIMERI OSTALIH GRAFOV

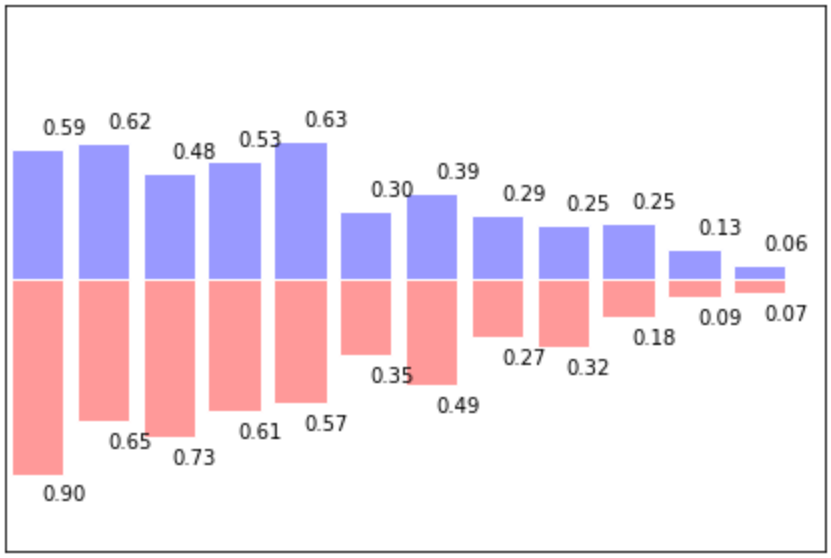
Scatter Plot

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

n = 1024
X = np.random.normal(0,1,n)
Y = np.random.normal(0,1,n)
T = np.arctan2(Y,X)

plt.axes([0.025,0.025,0.95,0.95])
plt.scatter(X,Y, s=75, c=T, alpha=.5)

plt.xlim(-1.5,1.5), plt.xticks([])
plt.ylim(-1.5,1.5), plt.yticks([])
# savefig('../figures/scatter_ex.png',dpi=48)
plt.show()
```



Bar plot

```
import numpy as np
import matplotlib.pyplot as plt
#%matplotlib notebook

n = 12
X = np.arange(n)
Y1 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)

plt.axes([0.025,0.025,0.95,0.95])
plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

for x,y in zip(X,Y1):
    plt.text(x+0.4, y+0.05, '%.2f' % y, ha='center', va= 'bottom')

for x,y in zip(X,Y2):
    plt.text(x+0.4, -y-0.05, '%.2f' % y, ha='center', va= 'top')

plt.xlim(-.5,n), plt.xticks([])
plt.ylim(-1.25,+1.25), plt.yticks([])

plt.show()
```



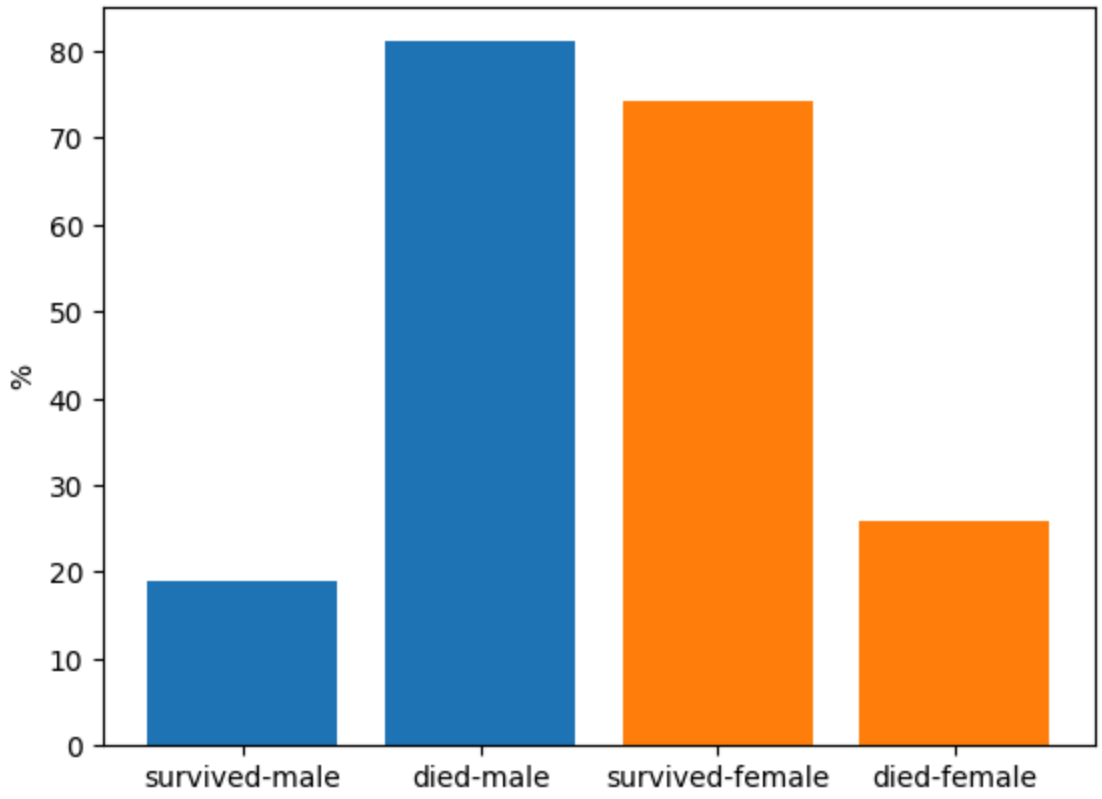
Pie chart

```
import numpy as np
import matplotlib.pyplot as plt
#%matplotlib notebook

n = 20
Z = np.ones(n)
Z[-1] *= 2

plt.axes([0.025,0.025,0.95,0.95])

plt.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i in range(n)])
plt.gca().set_aspect('equal')
plt.xticks([]), plt.yticks([])

plt.show()
```

Titanic Analysis

Sedaj bomo v stolpičnem grafu prikazali možnosti preživetja za oba spola.

```
import matplotlib.pyplot as plt
#%matplotlib notebook

columns = ["survived", "died"]
for s in survived_dist:
    x = [f"{c}-{s}" for c in columns]
    surv = survived_dist[s]["survived_%"] * 100
    died = survived_dist[s]["died_%"] * 100
    plt.bar(x, [surv, died],)

plt.ylabel("%")
plt.show()
```



Titanic analysis

Ponovimo analizo in tokrat poglejmo kakšna je bila možnost preživetja med različnimi starostnimi skupinami.

Skupine razdelimo v:

- 0 - 20
- 21 - 40
- 41 - 60
- 61 - 80
- 81 - 100

Za hitrejše programiranje bomo na začetku gledali le prvih 5 oseb. Nato pa bomo preprosto rekli, naj se program sprehodi čez vse osebe.

```
age_dist = {"0to20": {"survived": 0, "died": 0},
            "21to40": {"survived": 0, "died": 0},
            "41to60": {"survived": 0, "died": 0},
            "61to80": {"survived": 0, "died": 0},
            "81to100": {"survived": 0, "died": 0},
            }

for line in data[:5]:
    age = int(line[6])

    # set the dictionary key
    if age <= 20:
        key = "0to20"
    elif age <= 40:
        key = "21to40"
    elif age <= 60:
        key = "41to60"
    elif age <= 80:
        key = "61to80"
    else:
        key = "81to100"

    if int(line[1]): # person survived
        age_dist[key]["survived"] += 1
    else: # person died
        age_dist[key]["died"] += 1

age_dist
```

```
{'0to20': {'survived': 0, 'died': 0},
 '21to40': {'survived': 3, 'died': 2},
 '41to60': {'survived': 0, 'died': 0},
 '61to80': {'survived': 0, 'died': 0},
 '81to100': {'survived': 0, 'died': 0}}
```

Sedaj uporabimo program za vse osebe:

```
age_dist = {"0to20": {"survived": 0, "died": 0},
            "21to40": {"survived": 0, "died": 0},
            "41to60": {"survived": 0, "died": 0},
            "61to80": {"survived": 0, "died": 0},
            "81to100": {"survived": 0, "died": 0},
            }
```

```
for line in data: # Namesto 5 oseb bomo pregledali vse osebe
    age = int(line[6])

    # set the dictionary key
    if age <= 20:
        key = "0to20"
    elif age <= 40:
        key = "21to40"
    elif age <= 60:
        key = "41to60"
    elif age <= 80:
        key = "61to80"
    else:
        key = "81to100"

    if int(line[1]): # Oseba je preživela
        age_dist[key]["survived"] += 1
    else: # Oseba ni preživela
        age_dist[key]["died"] += 1

age_dist
```

Vidimo, da sedaj pride do errorja. Do errorja je prišlo, ker za določene osebe nimamo podatka o njihovi starost. In tam imamo prazen string `""`. Praznega string-a pa se ne da spremeniti v integer.

Errors

Errors so napake v programu, ki nam ponavadi zaustavijo izvajanje programa.

Klasificiramo jih v:

- Snytax 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

```
# Primer: manjka keyword def
myfunction(x, y):
    return x + y
```

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

```
else:
    print("Hello!")
```

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

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

```
  Cell In[2], line 2
    if mark >= 50
                 ^
SyntaxError: expected ':'
```

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

```
  Cell In[3], line 4
    esle:
         ^
SyntaxError: invalid syntax
```

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

```
  Cell In[4], line 3
    print("Flag is set!")
    ^
IndentationError: expected an indented block after 'if' statement on line 2
```

Runtime errors

Primer runtime errors:

- Deljenje z 0
- Dostopanje do elementov, ki ne obstajajo
- Dostopanje do datotek, ki ne obstajajo
- Izvajanje operacij na neskladnih tipih
- Uporaba nedefiniranega identifier-ja

```
# Primer: deljenje z 0
1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[5], line 2
      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.

The try and except statements

Da obvladujemo morebitne napake uporabljamo try-except:

```
for _ in range(3):
    x = int(input("Vnesi prvo številko: "))
    y = int(input("Vnesi drugo številko: "))
    rezultat = x / y
    # Ni preverjanja, če je:
    #    vnešen y slučajno 0
    #    kater od x ali y neštevilski znak
```

```
    print(f"{x}/{y} = {rezultat}")
    print()
```

Ko se zgodi napaka, Python preveri če se naša koda nahaja znotraj **try** bloka. Če se ne, 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.

```
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()
```

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

```
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()
```

```
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()
```

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
```

```
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())
```

Handling an error as an object

```
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 ravnamo z napako, ne dobimo nobenega povratne informacije o napaki. Ne vemo torej, zakaj je prišlo do napake in do kakšne napake je prišlo.

```
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

```
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()
```

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

Lahko pa različne errorje obravnavamo na različen način.

Preprosto dodamo še en except stavek.

```
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()
```

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 sesul.

Če se ujemata bo `except` error obravnaval. `Except` obravnava napake tega razreda in vse, ki dedujejo iz tega razreda.

Malo o izjemah

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

```
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))
```

Raising exceptions

Napake lahko napovemo / rais-amo tudi sami.

```
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()
```

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.

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
```

```
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()
```

The else and finally statements

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

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

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

print("End")
```

```
To ni številka.
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.

```
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.

Malo o ustvarjanju svojih exceptions: 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.

```
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**.

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.

```
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.
```

Titanic Analysis

Dopolnimo naš program tako, da če pride do napake tisto osebo preskočimo.

```
age_dist = {"0to20": {"survived": 0, "died": 0},
            "21to40": {"survived": 0, "died": 0},
            "41to60": {"survived": 0, "died": 0},
            "61to80": {"survived": 0, "died": 0},
            }


for line in data: # Namesto 5 oseb bomo pregledali vse osebe
    try:
        age = int(line[6])
    except ValueError:
        continue
    # set the dictionary key
    if age <= 20:
        key = "0to20"
    elif age <= 40:
        key = "21to40"
    elif age <= 60:
        key = "41to60"
    elif age <= 80:
        key = "61to80"
```

```
    if int(line[1]): # person survived
        age_dist[key]["survived"] += 1
    else: # person died
        age_dist[key]["died"] += 1

age_dist
```

```
{'0to20': {'survived': 75, 'died': 96},
 '21to40': {'survived': 152, 'died': 222},
 '41to60': {'survived': 50, 'died': 73},
 '61to80': {'survived': 5, 'died': 16}}
```

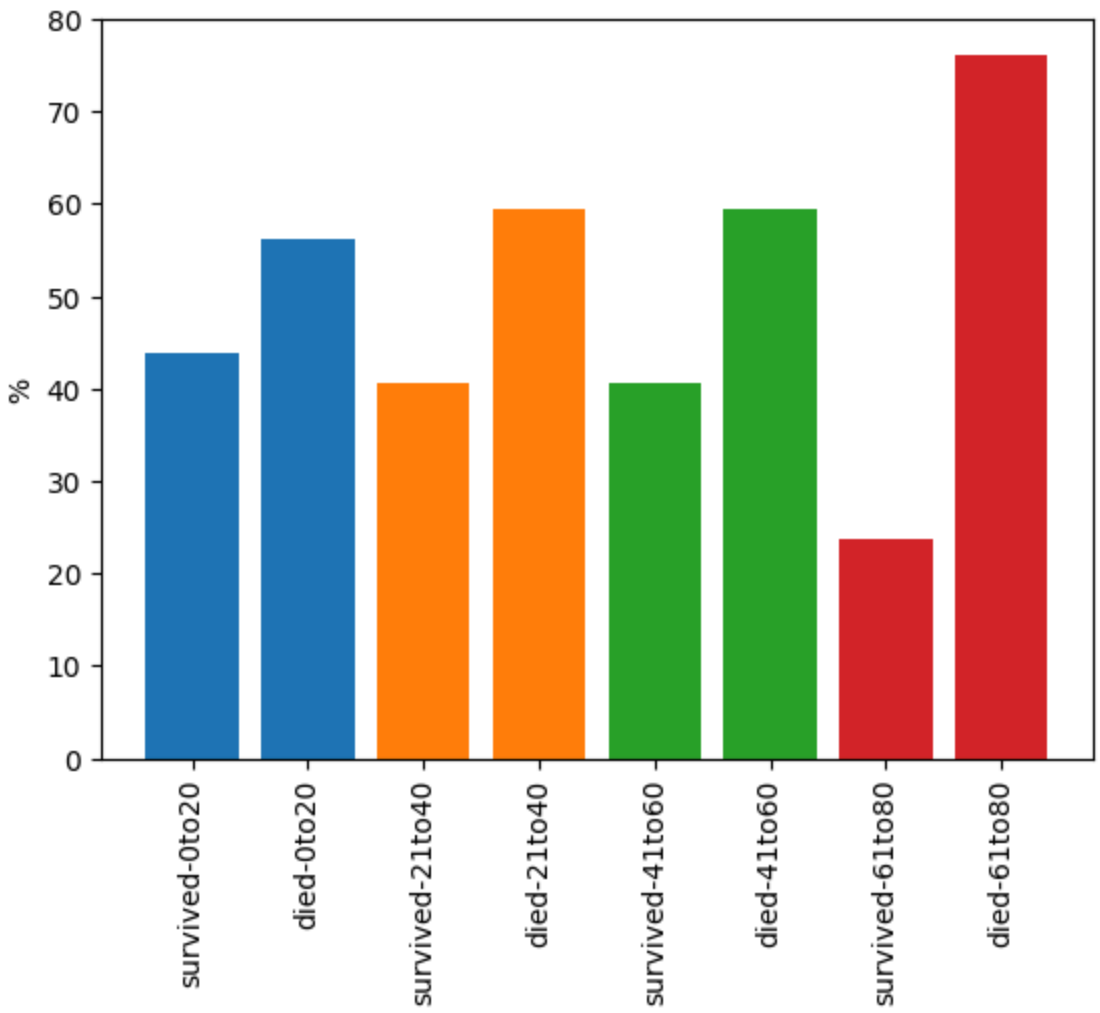
```
age_dist

for a in age_dist:
    total = age_dist[a]["survived"] + age_dist[a]["died"]
    age_dist[a]["survived_%"] = age_dist[a]["survived"] / total
    age_dist[a]["died_%"] = age_dist[a]["died"] / total

age_dist
```

```
{'0to20': {'survived': 75,
  'died': 96,
  'survived_%': 0.43859649122807015,
  'died_%': 0.5614035087719298},
 '21to40': {'survived': 152,
  'died': 222,
  'survived_%': 0.40641711229946526,
  'died_%': 0.5935828877005348},
 '41to60': {'survived': 50,
  'died': 73,
  'survived_%': 0.4065040650406504,
  'died_%': 0.5934959349593496},
 '61to80': {'survived': 5,
  'died': 16,
  'survived_%': 0.23809523809523808,
  'died_%': 0.7619047619047619}}
```

Dodajmo še graf:

```
import matplotlib.pyplot as plt
#%matplotlib notebook

ax1 = plt.subplot()

columns = ["survived", "died"]
x_labels = []

for a in age_dist:
    x_label = [f"{c}-{a}" for c in columns]
    x_labels.extend(x_label)

    surv = age_dist[a]["survived_%"] * 100
    died = age_dist[a]["died_%"] * 100
    ax1.bar(x_label, [surv, died])
```

```
ax1.set_ylabel("%")
ax1.set_xticks(range(len(x_labels)))
ax1.set_xticklabels(x_labels, rotation=90)
plt.show()
```



__name__ variable

Python ima posebno spremenljivko __name__. V Pythonu je posebna spremenljivka __name__ vgrajena spremenljivka, ki določa kontekst izvajanja datoteke (skripte). Omogoča, da skripta prepozna, ali se izvaja kot glavni program ali je uvožena kot modul.

> Ko se Python datoteka izvaja neposredno (npr. z python ime_datoteke.py), ima spremenljivka __name__ vrednost "__main__".

> Ko je datoteka uvožena kot modul v drugo datoteko (z import), bo __name__ vseboval ime modula (tj. ime datoteke brez pripone .py).

Če zaženemo naš modul direktno, bo spremenljivka enaka __main__.

m1.py

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

my_name()
```

```
__main__
```

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

```
import m1

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

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()
```

```
__main__
```

skripta.py

```
import m1

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

Assert

Malo o assertions: https://www.tutorialspoint.com/python/assertions_in_python.htm

`Assert` je ključna beseda v Pythonu, ki se uporablja za preverjanje pogojev med izvajanjem programa.

Če pogoj za assert ni resničen, se sproži izjema AssertionError.

Namenjena je predvsem za debugging – preverjanje, ali del programa deluje pravilno.

```python
x = 10
assert x > 5, "x mora biti večji od 5" # ne bo napake, ker je x > 5
```

```python
x = 3
assert x > 5, "x mora biti večji od 5"
# AssertionError: x mora biti večji od 5
```

Preverjanje veljavnosti podatkov:

```python
def deljenje(a, b):
    assert b != 0, "Deljenje z nič ni dovoljeno!"
    return a / b

print(deljenje(10, 2))  # 5.0
print(deljenje(10, 0))  # AssertionError: Deljenje z nič ni dovoljeno!
```

Testiranje s pomočjo `assert`:

```python
def kvadrat(x):
    return x ** 2

# Preverimo, ali funkcija vrača pravilne rezultate.
assert kvadrat(3) == 9, "Test ni uspel za x=3"
assert kvadrat(0) == 0, "Test ni uspel za x=0"
assert kvadrat(-4) == 16, "Test ni uspel za x=-4"
```

Nasveti in omejitve

- Uporabnost: assert se uporablja predvsem za notranje preverjanje med razvojem ali testiranjem kode.
- Ne v produkciji: Če Python zaženemo z optimizacijo (python -O), se assert izjave ignorirajo. Zato se assert ne uporablja za preverjanje uporabniških vhodov v produkcijskem okolju.
- Zamenjava v produkciji: Za preverjanje vhodov in izhodov uporabite standardne izjeme, kot so ValueError, TypeError ipd.