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 že vgrajeno funkcijo `open()` za odpiranje datotek.

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

In [ ]:
```
f = open("test.txt")    # open file in current directory
#f = open("C:/Python33/README.txt")  # specifying full path
```

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"

In [ ]:
```
f = open("test.txt")      # equivalent to 'r' or 'rt'
```

In [ ]:
```
f = open("test.txt",'r')  # write in text mode
print(type(f))
print(f)
```

In [ ]:
```
f = open("test.txt",'w')  # write in text mode
```

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

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

Zapiranje datotek

Ko končamo z našo operacijo moramo datoteko zapreti, ker tako sprostimo vire, ki so vezani na uporabo datoteke (spomin, procesorska moč, itd..).

In [ ]:
```
f = open("test.txt", "a")
# perform file operations

f.close()
```

> na Linuxu ne dela ta f.close(). Še kr lah spreminjam. Ko se python zapre bo počistu. Če pa maš program k laufa dolgo, potem je problem.

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**.

In [ ]:
```
try:
    f = open("test.txt", "a")
    # perform file operations
    raise ValueError
finally:
    f.close()

# če ta koda deluje, potem bi morali biti zmožni spreminjati datoteko tudi potem
```

Python with Context Managers

Isto stvar dosežemo z uporabo `with statement`.

In [25]:
```
with open("test.txt", "a") as f:
    pass
    # perform file operations
```

Branje datotek

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

(Imamo datoteko katere vsebina je: *Hello World!\nThis is my file.*)

In [28]:
```
with open("test.txt",'r') as f:
    file_data = f.read()    # read all data
    print(file_data)

#print(file_data)
#file_data
```

```
Hello world
This is my file
```

In [29]:
```
with open("test.txt", "r") as f:
    file_data = f.read(2) # read the first 2 data
    print(file_data)
    file_data = f.read(6) # read the next 6 data
    print(file_data)
    file_data = f.read() # reads till the end of the file
    print(file_data)
    file_data = f.read() # further reading returns empty string
    print(file_data)
```

```
He
llo wo
rld
This is my file
```

> We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.

Po datoteki se lahko tudi premikamo z uporabo `seek()` in `tell()` metode.

In [30]:
```
with open("test.txt", "r") as f:
    print(f.tell()) # get current position of file cursor in characters
    f.read(4) # read 4 bytes
    print(f.tell())
```

```
0
4
```

In [31]:
```
with open("test.txt", "r") as f:
    print(f.tell()) # get position in bytes

    reading = f.read(6) # read 6 bytes
    print(reading)
    print(f.tell()) # get new position in bytes

    f.seek(0) # move cursor to position 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 loop`.

In [32]:
```
with open("test.txt", "r") as f:
    for line in f:
        print(line) # The lines in file itself has a newline character '\n'.
```

```
Hello world

This is my file
```

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

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

In [33]:
```
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 listo preostalih linij v datoteki.

(če prov vidm `readlines()` prebere vrstice in postavi cursor na konc)

In [34]:
```
with open("test.txt","r") as f:
    list_of_lines = f.readlines()
    print(list_of_lines)
    print(list_of_lines[1])
```

```
['Hello world\n', 'This is my file\n']
This is my file
```

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

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
```

In [37]:
```
def funkcija(n):
    with open("naloga2.txt", "r") as f:
        data = f.readlines()
        for line in data[-n:]:
            print(line, end="")


funkcija(3)
```

```
line 7
line 8
line 9
```

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
```

In [9]:
```
def dictionary():
    beseda = input("Vnesi besedo: ")

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

dictionary()
```

```
Vnesi besedo: meow
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.

In [ ]:
```
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")

# This program will create a new file named 'test.txt' if it does not exist. If
# We must include the newline characters ourselves to distinguish different line
```

In [ ]:
```
with open("test.txt",'a') as f:
    f.write("We are adding another line.")
    x = f.write("And another one")
    #print(x) write() returns number of bytes we wrote

# this program will open a file and append what we want to the end
```

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

# this program will create a new file 'test2.txt' if it doesn't exist and write
# if the file exists it will trow an error
```

Importing

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

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

Da importiramo modul uporabimo besedo `import`.

```
import moj_modul
```

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

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

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

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

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

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

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

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

In [ ]:
```
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.

In [ ]:
```
import math

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

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

In [ ]:
```
import math

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

In [ ]:
```

```

Naloga:

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

In [6]:
```
# Rešitev
import math

math.log(144, 12)
```

Out[6]:
```
2.0
```

Importing our own module

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

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

moj_modul.py

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


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


moja_spremenljivka = 100
```

skripta.py

In [ ]:
```
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
```

In [ ]:
```
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
```

In [ ]:
```
from moj_modul import moja_spremenljivka

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

In [ ]:
```
from moj_modul import sestevalnik

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

In [ ]:
```
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
```

In [ ]:
```
import moj_modul as mm

print(dir())

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

print(mm.sestevalnik(5, 6))
```

```
print(mm.moja_spremenljivka)
```

In [ ]:
```
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

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

spremenljivka2 = 200
```

skripty.py

In [ ]:
```
from moj_package import modul2

print(dir())

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

In [ ]:
```
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.

In [ ]:
```
# 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.")
```

In [ ]:
```
# Rešitev
import math

def pretvornik(x ,mode="deg2rad"):
    if mode == "deg2rad":
        return math.radians(x)
    elif mode == "rad2deg":
        return math.degrees(x)
```

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

In [ ]:
```
from math import *

print(dir())
print(pi)

pi = 3
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
```

In [25]:
```
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()
```

JSON

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.

Če boste ustvarjali program, ki zajemajo podatke iz interneta, boste JSON velikokrat videli.

```
{
    "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.

In [2]:
```
import json
```

In [14]:
```
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.

In [15]:
```
json_data = json.dumps(data)
json_data
```

Out[15]:
```
'{"firstName": "Jane", "lastName": "Doe", "hobbies": ["running", "sky diving",
"singing"], "age": 35, "children": [{"firstName": "Alice", "age": 6}, {"firstNa
me": "Bob", "age": 8}]}'
```

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

In [16]:
```
py_data = json.loads(json_data)

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

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

Out[16]:
```
'Jane'
```

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

In [18]:
```
with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
# Note that dump() takes two positional arguments:
#(1) the data object to be serialized,
#and (2) the file-like object to which the bytes will be written.
```

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

Potrebno se je zavedati, da konverzija datatipov ni exactna (ni točna). To pomeni, če smo v začetnih podatkih imeli nekje tuple in ga shranili v JSON in ta JSON nato prebrali nazaj v python, tuple postane list.

Python-JSON conversion table

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

In [19]:
```
with open("data_file.json", "r") as read_file:
    data = json.load(read_file) # use loads() if the JSON data is in "python str
    print(data)
    print(type(data))
```

```
{'firstName': 'Jane', 'lastName': 'Doe', 'hobbies': ['running', 'sky diving',
'singing'], 'age': 35, 'children': [{'firstName': 'Alice', 'age': 6}, {'firstNa
me': '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€.

In [22]:
```
import json

with open("podatki.json") as f:
    data = json.load(f) # use loads() if the JSON data is in "python string" typ
    #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']
```

In [47]:
```
data = []

with open("./titanic.csv", "r") as f:
    for line in f.readlines()[1:]: # preskočimo prvo vrstico, ker so to imena st
        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 21
171', '7.25', '', 'S']
['2', '1', '1', '"Cumings', ' Mrs. John Bradley (Florence Briggs Thayer)"', 'fe
male', '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', '37345
0', '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}}
```

In [75]:
```
survived_dist = {"male": {"survived": 0, "died": 0}, "female": {"survived": 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
```

Out[75]:
```
{'male': {'survived': 109, 'died': 468},
 'female': {'survived': 233, 'died': 81}}
```

In [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
```

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

Sedaj bi želeli to grafično prikazati.

Da pa ne bomo sami pisali celotne kode za grafični prikat in izdelavo grafov, bomo uporabili že narejeno knjižnjico.

3rd party libraries

Zgoraj smo importirali kodo, ki je prišla skupaj z inštalacijo Python-a, oziroma kodo katero smo spisali sami.

Kodo katero 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)
```

Če dobimo napako, moramo pip inštalirati:

> 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:

`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

In [1]:
```
#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` is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Simple plot

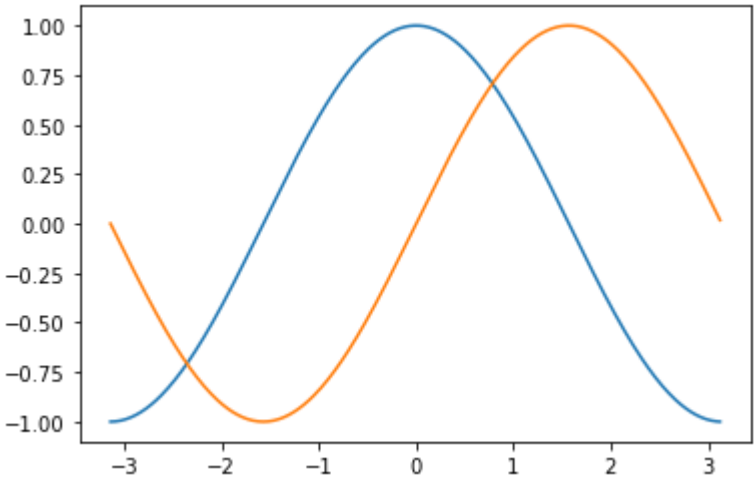
Začeli bomo s preprostim primerom:

> **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.

In [28]:
```
import matplotlib.pyplot as plt
import math
#x-os
x = [math.radians(x) for x in range(-180, 180)]

#y-os, potrebujemo vrednsoti 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.

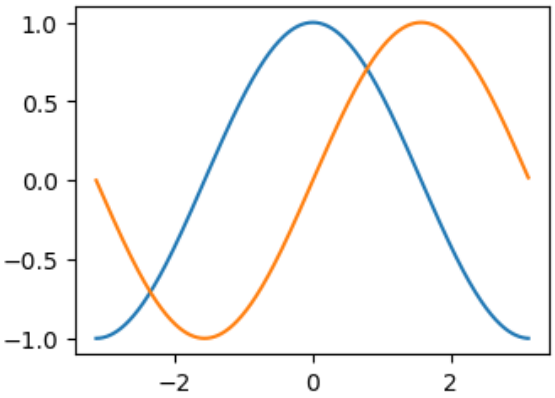
Figure v matplotlib-u pomeni celotno okno. Za naslov imajo `Figure _cifra_`, številčenje pa se začne z 1. Ostali parametri, ki definirajo `figure` so:

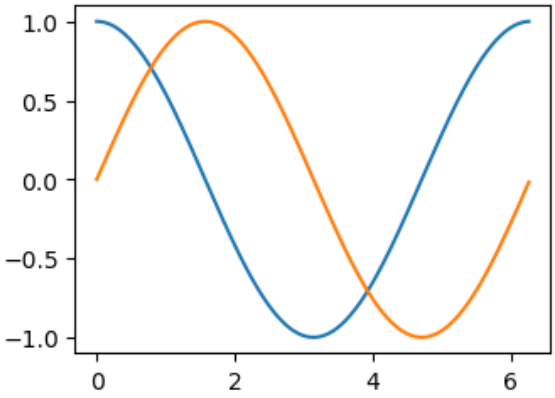
- figsize - velikost figure v inčih (dolžina, višina)
- dpi - resolucija v dots per inch
- facecolor - barva risalne podlage
- edgecolor - barva obrobe

Subplot

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

In [29]:
```
import matplotlib.pyplot as plt
import math

# Create a new figure of size 8x6 points, using 100 dots per inch
plt.figure(figsize=(8,6), dpi=100)

# Create a new subplot from a grid of 1x1
# 2 rows, 2 columns, and the 2nd subplot
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)




# Show result on screen
plt.show()
```

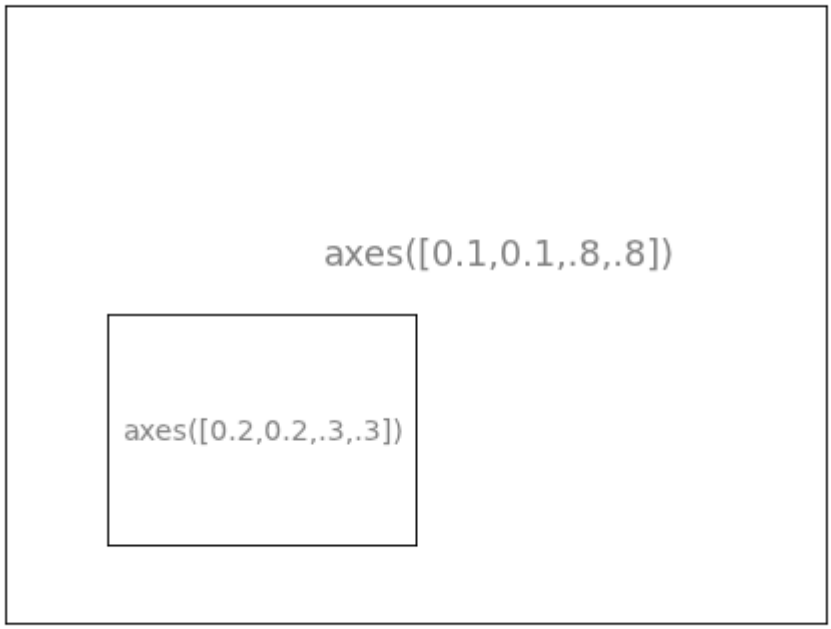




Axes se obnašajo podobno kot `subplot`, le da oni lahko ležijo kjerkoli v figuri.

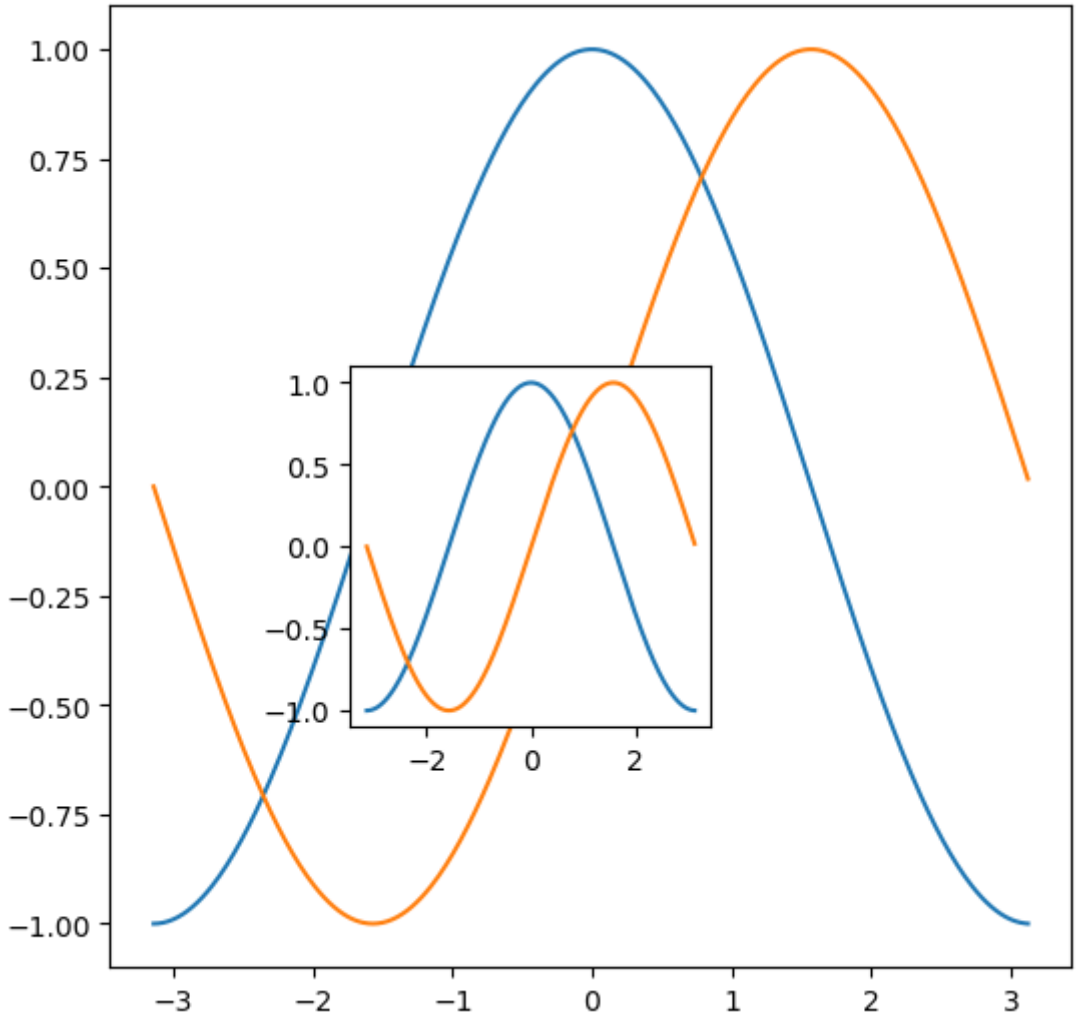
`axes([left, bottom, width, height])`

Width in height sta normalizirana glede na figuro.



In [38]:
```
#import matplotlib.pyplot as plt
#import math

# Create a new figure of size 8x6 points, using 100 dots per inch
```

```
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]


# Create a new subplot from a grid of 1x1
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)

# Show result on screen
plt.show()
```

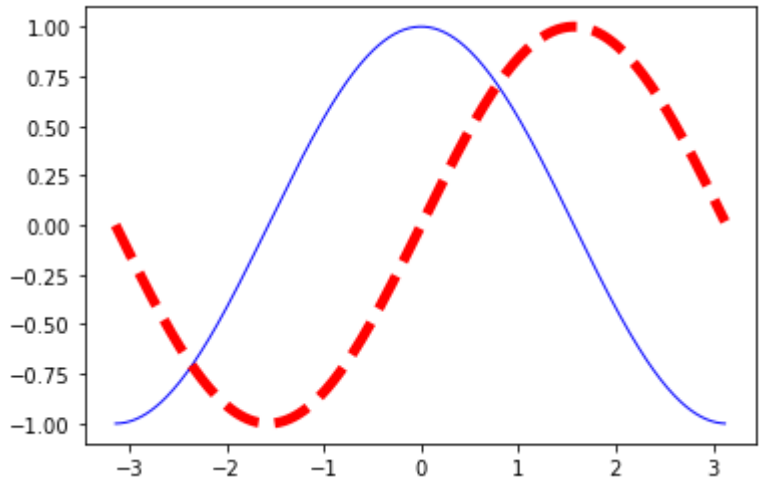


Spreminjanje lastnosti

Spreminjanje lastnosti črt

In [39]:
```
#import matplotlib.pyplot as plt
#import math
```

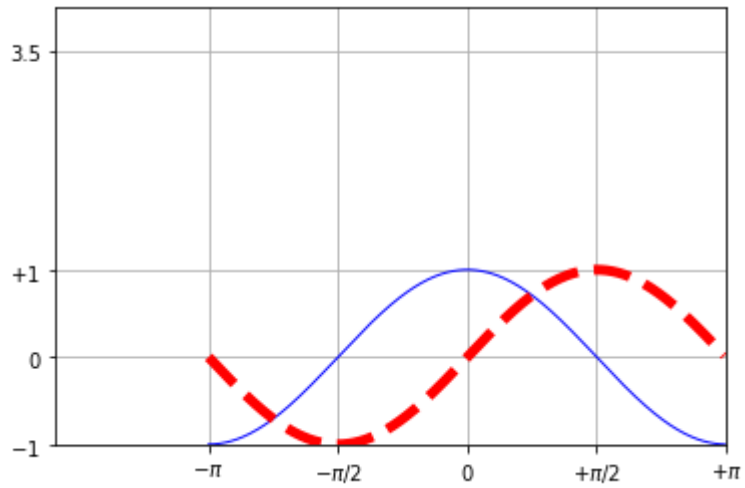
```
X = [math.radians(x) for x in range(-180, 180)]
C = [math.cos(i) for i in X]
S = [math.sin(i) for i in X]

# 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

In [40]:
```
#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
# 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

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

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



plt.show()
```

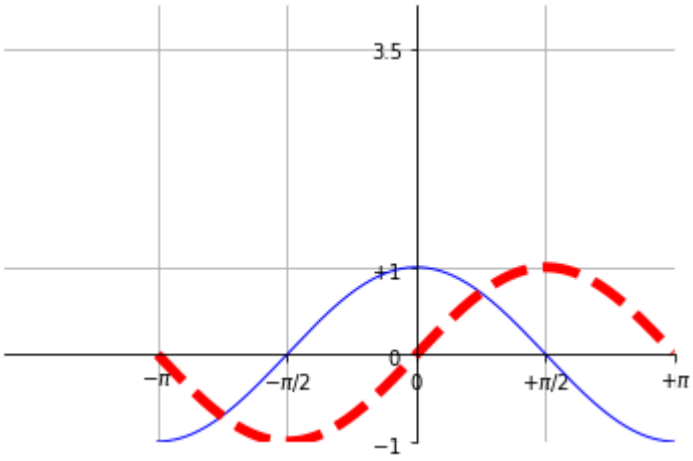


Spreminjanje lastnosti okvira

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

In [44]:
```
#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
```

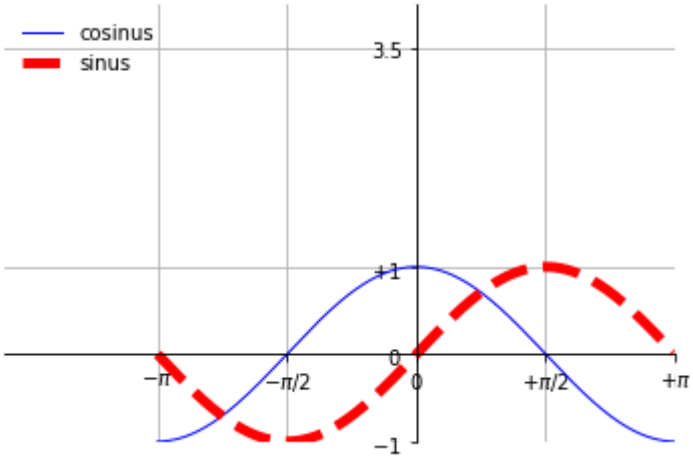
```
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
#Plotting cosinus funkcijo
# color="blue" -> črta bo modre barve
# linewidh=1.0 -> debelina črte bo 1pixel
# linestyle="-" -> oblika črte bo neprekinjena

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
# 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

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

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
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 n
ax.spines['left'].set_position(('data',0))

plt.show()
```



Dodajanje legende

In [45]:
```
#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
# 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

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
```

```
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
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 n
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: "la
# loc -> lokacija
# frameon -> če je uokvirjena legenda ali ne

plt.show()
```

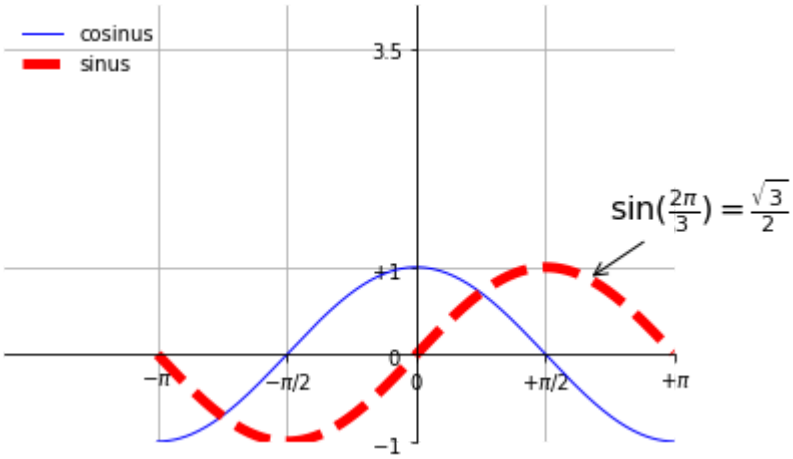


Anotacija

In [47]:
```
#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
# 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

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

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
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 n
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: "la
# 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 p
#xycoords="data" -> koordinatni sistem v katerem sta xy=
#xytext -> specificiram kam naj vstavi text
#textcoords -> koordinatni sistem za text -> 'offset points'    offset (in point
#arrowprops
#   arrowstyle -> kakšnega stila je naša puščica



plt.show()
```



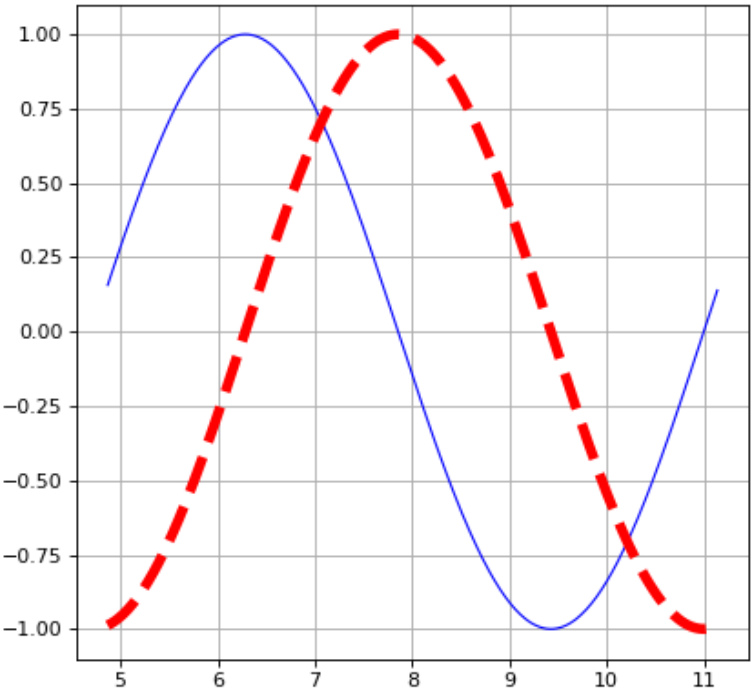
Animation

`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

In [48]:
```
#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 približ
plt.show()
```

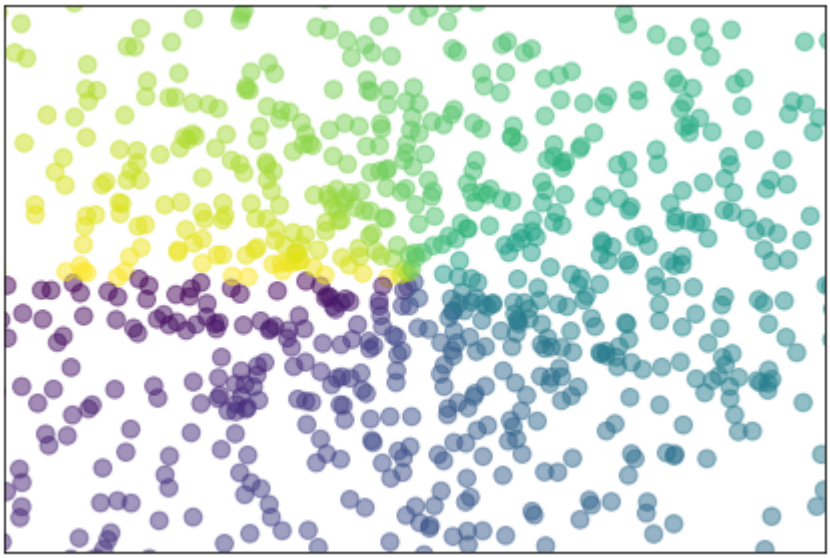


PRIMERI OSTALIH GRAFOV

Scatter Plot

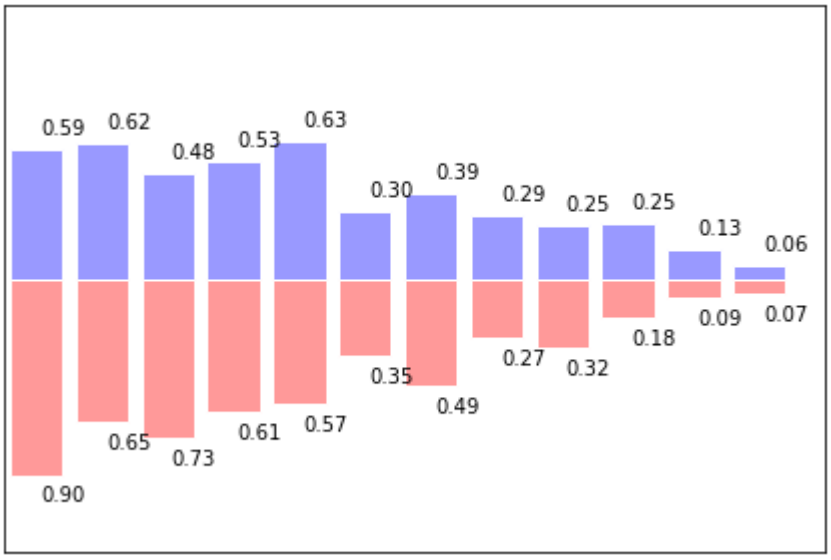
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

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

In [2]:
```
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

In [3]:
```
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()
```

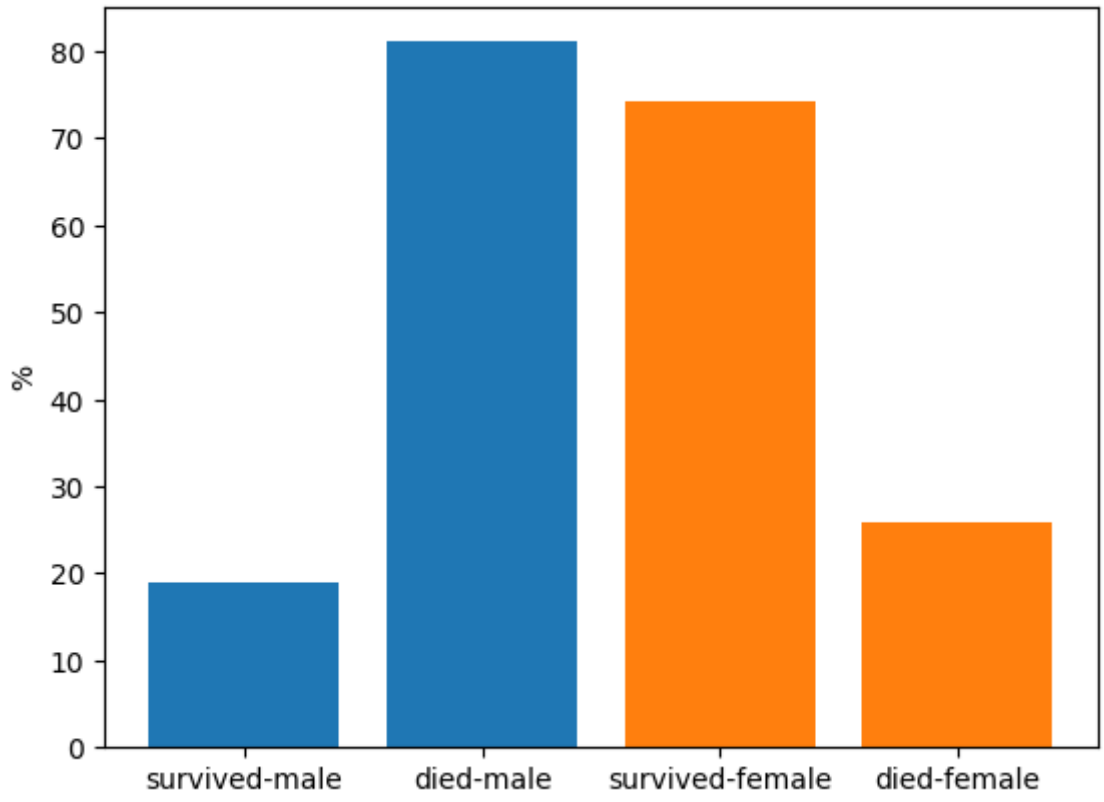
Titanic Analysis

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

In [83]:
```
survived_dist
```

Out[83]:
```
{'male': {'survived': 109,
  'died': 468,
  'survived_%': 0.18890814558058924,
  'died_%': 0.8110918544194108},
 'female': {'survived': 233,
  'died': 81,
  'survived_%': 0.7420382165605095,
  'died_%': 0.25796178343949044}}
```

In [112…
```
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.

In [108…
```
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
```

Out[108]:
```
{'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:

In [109…
```
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]): # person survived
        age_dist[key]["survived"] += 1
    else: # person died
        age_dist[key]["died"] += 1

age_dist
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [109], line 10
      1 age_dist = {"0to20": {"survived": 0, "died": 0},
      2              "21to40": {"survived": 0, "died": 0},
      3              "41to60": {"survived": 0, "died": 0},
      4              "61to80": {"survived": 0, "died": 0},
      5              "81to100": {"survived": 0, "died": 0},
      6              }
      9 for line in data: # Namesto 5 oseb bomo pregledali vse osebe
---> 10     age = int(line[6])
     12     # set the dictionary key
     13     if age <= 20:

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

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

Problem bi lahko rešili na več načinov, ampak poglejmo si kako se v pythonu lotimo obvladovanja errorjev.

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.

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
```

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!
```

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
```

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)

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 število: 1
Vnesi drugo število: 0
Zmeraj ta prestreže.

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

Vnesi prvo število: 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'>)
```

Raising exceptions

Napake lahko rais-amo tudi sami.

In [18]:
```
def delilnik_pozitivnih_st():
    try:
        x = int(input("Vnesi prvo število: "))
        if x < 0:
            raise ValueError("Vnešena mora biti pozitivna številka")

        y = int(input("Vnesi drugo število: "))
        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 število: -1
Vnešena mora biti pozitivna številka

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

Vnesi prvo število: 1
Vnesi drugo število: 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.

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()
```

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.

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**.

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.
```

Titanic Analysis

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

In [120…
```
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
```

Out[120]:
```
{'0to20': {'survived': 75, 'died': 96},
 '21to40': {'survived': 152, 'died': 222},
 '41to60': {'survived': 50, 'died': 73},
 '61to80': {'survived': 5, 'died': 16}}
```

In [121…
```
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
```

Out[121]:
```
{'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:

In [134…

```
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()
```