In [1]:
```
%%html
<style>
.not_clean {
    background: lightgreen;
    padding: 20px;
}

.comment {
    background: #F3FF70;
    padding: 20px;
}
</style>
```

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

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`

JSON	Python
`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€.
```

In [ ]: