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 [ ]:

In [ ]: