

In [1]:

```
versoin
```

```
-----  
-  
NameError                                Traceback (most recent call las  
t)  
<ipython-input-1-0da2a7b0c4cb> in <module>  
----> 1 versoin  
  
NameError: name 'versoin' is not defined
```

In [30]:

```
print("tanoj")
```

tanoj

In [31]:

```
age = 20  
print('age')
```

20

In [32]:

```
print(age)
```

20

In [7]:

```
first_name = "Tanoj"  
last_name = "N"  
full_name = f"{first_name} {last_name}"  
print(full_name)
```

Tanoj N

In [8]:

```
## addressing whitespaces
```

In [9]:

```
print("Python")
```

Python

In [12]:

```
print("\tpython") # print using tab space
```

python

In [13]:

```
print("\nPython") # print in new line
```

Python

In [14]:

```
print("Languages:\nPython\nScala\nJava")
```

Languages:

Python

Scala

Java

In [17]:

```
x = "value"
```

In [19]:

```
x = "value "
```

```
print(x)
```

value

In [20]:

```
print(x.strip()) # removes the white spaces
```

value

In [21]:

```
# Numbers
```

In [22]:

```
2+1
```

Out[22]:

3

In [23]:

```
3/2
```

Out[23]:

1.5

In [24]:

```
x=1
```

```
y=2
```

```
z=3
```

In [26]:

```
x,y,z=1,2,3 # advisable to declare a variable like this when we need to assign more variables
```

In [27]:

```
import this
```

The Zen of Python, by Tim Peters

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

In [33]:

```
import sys
```

In [1]:

```
#List -collection of items in a particular and it is mutable data type.
#String is an immutable data type

#it can be assigned by using []
```

In [10]:

```
bicycles = ['trek','redline','hero']
# elements will be accessed with 0 by indexing eg 0,1,2,3,...

print(bicycles[0])           # here to print 1st element
print(bicycles[0].title())   # capital first letter
```

```
trek
Trek
```

In [18]:

```
# assigning a new value i.e. change trek to ranger  
  
bicycles[2] = 'ranger' # if we replace 0 with 2 in the 2nd index place  
print(bicycles)  
  
['ranger', 'redline', 'ranger']
```

In [19]:

```
#insert and apend  
print(bicycles)  
  
['ranger', 'redline', 'ranger']
```

In [20]:

```
bicycles[2] = 'hero'  
bicycles[0] = 'trek'  
print(bicycles)  
  
['trek', 'redline', 'hero']
```

In [22]:

```
bicycles.append('ranger') # here it appends ranger to 4th position  
print(bicycles)  
  
['trek', 'redline', 'hero', 'ranger']
```

In [23]:

```
#include the element to designed index  
#to achive this use 'insert'  
bicycles.insert(1,'ranger') #index,value  
print(bicycles)  
  
['trek', 'ranger', 'redline', 'hero', 'ranger']
```

In [24]:

```
#how to delete a value from list  
# pop() : to delete  
  
bicycles.pop()  
print(bicycles)  
  
['trek', 'ranger', 'redline', 'hero']
```

In [25]:

```
bicycles.pop()  
print(bicycles) # from the above outputs pop() deletes only the LAST value  
  
['trek', 'ranger', 'redline']
```

In [27]:

```
bicycles.pop(0)
print(bicycles)    #when we provide index value, it deletes the desired value

['ranger', 'redline']
```

In [31]:

```
cars = ['bmw', 'audi', 'jaugur', 'benz']

#sorting a list -- using sort()

cars.sort()          # default sort
cars.sort(reverse = True) # reverse sort
print(cars)

['jaugur', 'bmw', 'benz', 'audi']
```

In []: