

In [30]:

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

tanoj

In []:

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

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 va  
riables
```

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 mutuble daya type.  
#String is an immutable data type  
  
#it can be assigned by using []
```

In [10]:

```
bicycles = ['trek','redline','hero']  
# elements will be accessing 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 append  
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 [36]:

```
cars = ['bmw','audi','jaugur','benz']  
  
#sorting a list -- using sort()  
  
cars.sort()           # default sort  
cars.reverse()        # reverse sort .. we can also use "sort(reverse = True)"  
print(cars)  
print('cars:',cars )
```

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

In [37]:

```
# create an empty list
```

```
bikes = []  
print("ini")  
print(bikes)
```

```
ini  
[]
```

In [39]:

```
bikes = [ 'a','b']  
print(bikes)
```

```
['a', 'b']
```

In []:

```
# Negative indexing
```

```
indexing is 0,1,2,3,4...  
if there are 1000 elements and the 1000 th element will be indexed as -1  
from last to first -1,-2,-3,-4.....
```

In [42]:

```
cars = ['audi','bmw','bugatti','ferrari']  
print(cars[-1])           #-ve indexing
```

```
ferrari
```

In [45]:

```
# For Loops in Lists

Students = ['I', 'Me', 'Myself']
for student in Students:
    print(student)          # student is a temp variable
                           # space before print is called INDENTATION(maintaining a pr
oper format). It takes 4 spaces
```

I
Me
Myself

In [49]:

```
Students = ['I', 'Me', 'Myself']
for y in Students:
    print(f"{y.title()}, attendant of python")
#Eg: getting a bday wishes from ol stores. Message will be given and for will be names
```

I, attendant of python
Me, attendant of python
Myself, attendant of python

In [50]:

```
Students = ['I', 'Me', 'Myself']
for a in Students:
    print(f"{a.title()}, attendant of python") # f string

print("Thnq for participation") # this print is out of for loop
```

I, attendant of python
Me, attendant of python
Myself, attendant of python
Thnq for participation

In [57]:

```
Android = ['One+', 'Moto']
IOS = ['Apple']
for phone in Android && IOS:
    print(f"{phone.title()}, is Android")
    print(f"{phone.title()}, is IOS")

print("Are popular brands")
```

File "<ipython-input-57-e87349e9267e>", line 3

```
    for phone in Android && IOS:
                          ^
```

SyntaxError: invalid syntax

In [53]:

```
Students = ['I', 'Me', 'Myself']  
for a in Students  
    print(f"{a.title()}, attendant of python")
```

File "<ipython-input-53-f5dc8d216456>", line 2
 for a in Students
 ^

SyntaxError: invalid syntax

In [54]:

```
Students = ['I', 'Me', 'Myself']  
for a in Students:  
    print(f"{a.title()}, attendant of python")
```

File "<ipython-input-54-fc9c72351e11>", line 3
 print(f"{a.title()}, attendant of python")
 ^

IndentationError: expected an indented block

In [58]:

```
# working with numerical list  
  
numbers = list(range(1,6))    #RANGE -- inbuilt function to give the range  
print(numbers)
```

[1, 2, 3, 4, 5]

In [63]:

```
for value in range(1,11):  
    print(value)
```

1
2
3
4
5
6
7
8
9
10

In [72]:

```
squares = []    # create an empty list  
for value in range(1,16):  
    square = value**2    # * multiplies once and ** multiplies twice  
    squares.append(square)  
  
print(squares)
```

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225]

In []: