LAB MANUAL

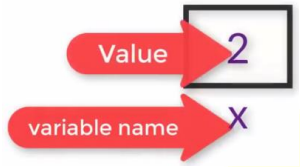
Python

2015 E.C

ETHIOP COMPUTING

Start with python console

Variables




```
>>> 2 + 3
5
>>> x = 2
>>> x + 3
5
>>> y = 3
>>> x + y
5
>>> x = 9
>>> x + y
12
>>> x
9
```

```
>>> x + 10
19
>>> _ + y
22
```

the "_" sign indicates the previous out put

```
>>> name = 'youtube'
>>> name
'youtube'
>>> name + ' rocks'
'youtube rocks'
>>>
>>> name ' rocks'
SyntaxError: invalid syntax
>>> name[0]
'y'
>>> name[6]
'e'
```


```
>>> name[8]
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    name[8]
IndexError: string index out of range
```

Why error occur? Because the variable "name" has only 7 characters. Hence name [8] is out of indexes.

```
>>> name[-1]
'e'
>>> name[-2]
'b'
>>> name[-7]
'y'
>>> name[0:2]
'yo'
>>> name[1:4]
'out'
>>> name[1:]
'outube'
>>> name[:4]
'yout'
```

```
>>> 'my ' + name[3:]
'my tube'
```

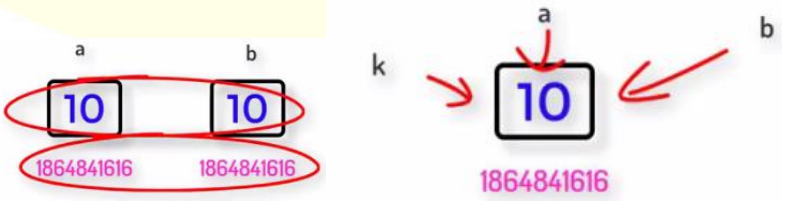
we can add string to existing string variables

```
>>> myname = 'Navin Reddy'
>>> len(myname)
11
```

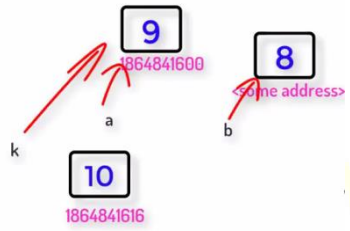
we can use this syntax to know the length of characters

More on Variables

```
>>> num = 5
>>> id(num)
1864841536
>>> name = 'navin'
>>> id(name)
90293952
>>> a = 10
>>> b = a
>>> a
10
>>> b
10
>>> id(a)
1864841616
>>> id(b)
1864841616
>>> id(10)
1864841616
>>> k = 10
>>> id(k)
1864841616
```



```
>>> a = 9
>>> id(a)
1864841600
>>> id(b)
1864841616
>>> k = a
>>> id(k)
1864841600
>>> b = 8
```




```
>>> PI = 3.14
>>> PI
3.14
>>> PI = 3.15
>>> type(PI)
<class 'float'>
```

We can call the declaration of PI inbuilt functions

Mathematical operation

Arithmetic Operators

```
IDLE Shell 3.11.0
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.0 (main, Oct 24 2022, 18:26:
Type "help", "copyright", "credits" or "
>>> 2+5
7
>>> 9-8
1
>>> 5*6
30
>>> 8/3
2.6666666666666665
>>> 8+2-7
3
```

Assignment Operators

```
>>> x = x + 2
>>> x
4
>>> x += 2
>>> x
6
>>> x *= 3
>>> x
18
>>> a,b = 5,6
>>> a
5
>>> b
6
```

Unary Operator

```
>>> n = 7
>>> n
7
>>> -n
-7
>>> n
7
>>> n = -n
>>> n
-7
```

Relational Operators

```
>>> a < b
True
>>> a > b
False
>>> a == b
False
>>> a = 6
>>> a == b
True
```

Logical Operators

```
>>> a = 5
>>> b = 4
>>> a < 8 and b < 5
True
>>> a < 8 and b<2
False
>>> a < 8 or b<2
True
```

Bitwise operator

- Complement (~)
- And (&)
- Or (|)
- XOR (^)
- Left Shift (<<)
- Right Shift (>>)

```
>>> ~12
-13
>>> 12 & 13
12
>>> 12 | 13
13
>>> 25 & 30
24
>>> 12 ^ 1
13
3
>>> 12 ^ 13
1
>>> 25 ^ 30
7
>>> 10 << 2
40
```

Import Mathematical function in python

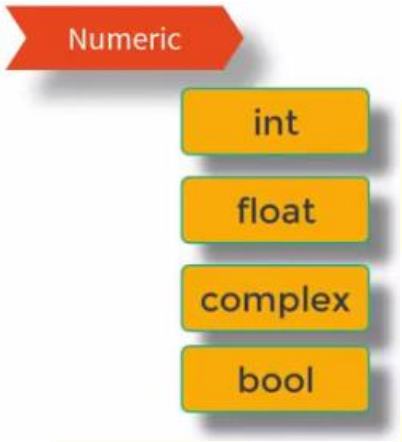
```
>>> import math
>>> x = math.sqrt(25)
>>> x
5.0
>>> x = math.sqrt(15)
>>> x
3.872983346207417
>>> print(math.floor(2.9))
2
>>> print(math.ceil(2.2))
3
```

```
>>> print(math.pow(3,2))
9.0
>>> print(math.pi)
3.141592653589793
>>> print(math.e)
2.718281828459045
```

```
>>> from math import sqrt, pow
>>> pow(4,5)
1024.0
```

Data Types

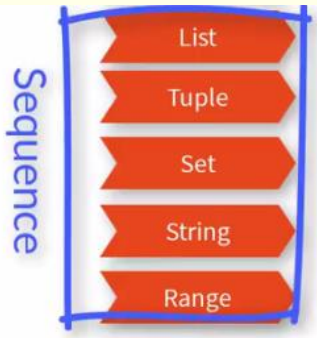
The common data types in Python are the following:




```
>>> num = 2.5
>>> type(num)
<class 'float'>
>>> num = 5
>>> type(num)
<class 'int'>
>>> num = 6+9j
>>> type(num)
<class 'complex'>
>>> a = 5.6
>>> b = int(a)
>>> type(b)
<class 'int'>
>>> b
5
>>> k = float(b)
>>> k
5.0
>>> k = 6
>>> c = complex(b,k)
>>> c
(5+6j)
```

```
>>> b<k
True
>>> bool = b < k
>>> bool
True
>>> type(bool)
<class 'bool'>
>>> b > k
False
>>> int(True)
1
>>> int(False)
0
```

Sequences

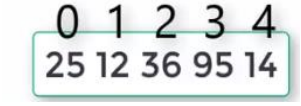



```
>>> lst = [25,36,45,12]
>>> type(lst)
<class 'list'>
>>> s = {25,36,45,15,12,25}
>>> s
{36, 12, 45, 15, 25}
>>> type(s)
<class 'set'>
>>> t = (25,36,4,57,12)
>>> type(t)
<class 'tuple'>
>>> str = "navin"
>>> st = 'a'
\
>>> type(st)
<class 'str'>
>>> range(10)
range(0, 10)
```

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,10,2))
[2, 4, 6, 8]
>>> type(range(10))
<class 'range'>
```

List

Operation	Result	Notes
"s[i] = x"	item *i* of *s* is replaced by *x*	
"s[i:j] = t"	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*	
"del s[i:j]"	same as "s[i:j] = []"	
"s[i:j:k] = t"	the elements of "s[i:j:k]" are replaced by those of *t*	(1)
"del s[i:j:k]"	removes the elements of "s[i:j:k]" from the list	
"s.append(x)"	appends *x* to the end of the sequence (same as "s[len(s):len(s)] = [x]")	
"s.clear()"	removes all items from *s* (same as "del s[:]")	(5)
"s.copy()"	creates a shallow copy of *s* (same as "s[:]")	(5)
"s.extend(t)" or "s += t"	extends *s* with the contents of *t* (for the most part the same as "s[len(s):len(s)] = t")	
"s *= n"	updates *s* with its contents repeated *n* times	(6)
"s.insert(i, x)"	inserts *x* into *s* at the index given by *i* (same as "s[i:i] = [x]")	
"s.pop()" or "s.pop(i)"	retrieves the item at *i* and also removes it from *s*	(2)
"s.remove(x)"	remove the first item from *s* where "s[i]" is equal to *x*	(3)




```
>>> num=[25,12,36,95,14]
>>> num
[25, 12, 36, 95, 14]
>>> num[0]
25
>>> num[4]
14
>>> num[6]
...
SyntaxError: unexpected indent
>>> num[6]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    num[6]
IndexError: list index out of range
```

Here in list, we can not specify out of index range. We have to also sure line indentation. Indentation should be correct in all python code

```
>>> nums[2:]

[36, 95, 14]
>>> nums[-1]

14
>>> nums[-5]

25
>>> names = ['navin','kiran','john']

>>> names

['navin', 'kiran', 'john']
>>> values = [9.5,'Navin',25]

>>> mil = [nums, names]

>>> mil

[[25, 12, 36, 95, 14], ['navin', 'kiran', 'john']]
```

Remember List is mutable. Hence, we can Insert, remove and delete value from list

```
>>> nums

[25, 12, 36, 95, 14, 45]
>>> nums.insert(2,77)

>>> nums

[25, 12, 77, 36, 95, 14, 45]
>>> nums.remove(14)

>>> nums

[25, 12, 77, 36, 95, 45]
>>> nums.pop(1)

12
>>> nums

[25, 77, 36, 95, 45]
>>> nums.pop()

45
```

```
>>> del nums[2:]

>>> nums

[25, 77]
>>> nums.extend(29,12,14,36)

Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    nums.extend(29,12,14,36)
TypeError: extend() takes exactly one argument (4 given)
>>> nums.extend([29,12,14,36])

>>> nums

[25, 77, 29, 12, 14, 36]
```

Mathimatical operation in list

```
>>> min(nums)

12
>>> max(nums)

77
>>> sum(nums)

193
>>> nums.sort()

>>> nums

[12, 14, 25, 29, 36, 77]
>>>
```

Tuple

We cannot change the value of tuple.

```
>>> tup = (21,36,14,25)
>>> tup
(21, 36, 14, 25)
>>> tup[1]
36
>>> tup[1] = 33
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tup[1] = 33
TypeError: 'tuple' object does not support item assignment
```

this error is due to the inability of changing value in tuple

Set

A collection of unique elements

```
>>> s = {22,25,14,21,5}
>>> s
{5, 14, 21, 22, 25}
>>> s = {25,14,98,63,75,98}
>>> s
{98, 75, 14, 25, 63}
>>> s[2]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    s[2]
TypeError: 'set' object does not support indexing
```

Set does not support indexing. Because its value does not keep sequence

Number Conversions

```
>>> bin(25)
'0b11001'
>>> 0b0101
5
>>> oct(25)
'0o31'
>>> hex(25)
'0x19'
>>> hex(10)
'0xa'
>>> 0xf
15
```

Starting with PyCharm

Get input from user

main.py | myfirst.py

```
x=input("enter the 1st number")
y=input("enter the 2nd number")
x=int(x)
y=int(y)
z=x+y
print(z)
```

or

main.py | myfirst.py

```
x=int(input("enter the 1st number"))
y=int(input("enter the 2nd number"))
z=x+y
print(z)
```

In both cases the output will be as follows

```
enter the 1st number6
enter the 2nd number6
12

Process finished with exit code 0
```

main.py | myfirst.py

```
ch=input('enter a character')[0]
print(ch)
```

```
enter a characterccharacter
c

Process finished with exit code 0
```

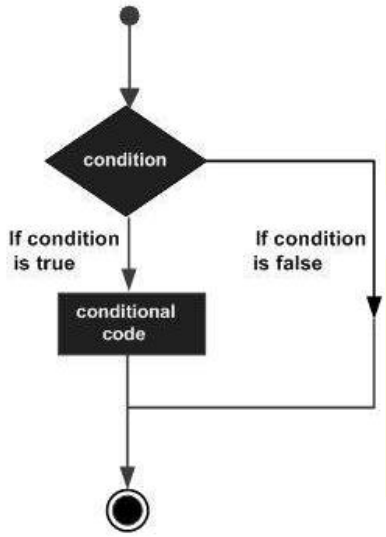
User enter the character “character”. But the output is only “c”. because the user limits the code to print only one character.

Python - Decision Making

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision-making structure found in most of the programming languages.



Sr.No.	Statement & Description
1	if statements An **if statement** consists of a boolean expression followed by one or more statements.
2	if...else statements An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
3	nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

If statement

main.py | myfirst.py | if_statement.py

```
x=10
r=x%2
if r==0:
    print('X is even')
if r==1:
    print('X is odd')
```

```
C:\Users\Misganaw\PycharmProjects\m
X is even

Process finished with exit code 0
```

```
x=10
r=x%2
if r==0:
    print('X is even')
if r==1:
    print('X is odd')
y=34
if y%2==0:
    print('y is even')
else:
    print('y is odd')
```

```
C:\Users\Misganaw\PycharmProjects\my
X is even
y is even

Process finished with exit code 0
```

Cascaded if else

cascaded_ie_else.py

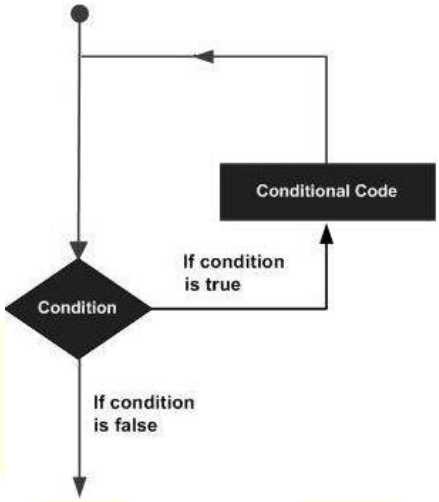
```
x=10
k=x/2
r=x%2
y=6
if r==0:
    print('X is even')
    if r>y:
        print('X is greater than y')
    else:
        print('X is less than Y')
else:
    print('y is odd')
```

```
C:\Users\Misganaw\PycharmProjects\my
X is even
X is less than Y

Process finished with exit code 0
```

Python – Loops

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement.



Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	nested loops You can use one or more loop inside any another while, for or do..while loop.

For loop

```
x='python tutorial'
for i in x:
    print(i,end=' ')
```

```
p y t h o n   t u t o r i a l
```

```
x='python tutorial'
for i in x:
    print(i,end=' ')
for i in range(5):
    print(i)
```

```
p y t h o n   t u t o r i a l 0
1
2
3
4
```

```
for i in range(11,15):
    print(i)
```

```
11
12
13
14
```

```
for i in range(11,15,2):
    print(i)
```

```
11
13
```

```
for i in range(20,12,-2):
    print(i)
```

```
20
18
16
14
```

```
for i in range(1,16):
    if(i%2!=0):
        print(i)
```

```
1
3
5
7
9
11
13
15
```

While loop

```
i=1
while i<=5:
    print('echo',end=' ')
    j=1
    while j<=5:
        print ('print J',end=' ')
        j=j+1
    i=i+1
    print()
```

```
echo print J print J print J print J print J
echo print J print J print J print J print J
echo print J print J print J print J print J
echo print J print J print J print J print J
echo print J print J print J print J print J
```

Unconditional Control structure

Break statement

```
num=6
x=int(input('Hi programmer enter the number'))
i=1
while i<=x:
    if i>num:
        print('out of stuck')
        break;
    print('hero programmer')
    i+=1
```

```
Hi programmer enter the number10
hero programmer
hero programmer
hero programmer
hero programmer
hero programmer
hero programmer
out of stuck
```

Continue statement

```
i=0
for i in range(1,10):
    if i%2==0:
        print('is is divisible by 2. so, i is not displyed')
        continue
    print(i)
    i+=1
```

```
1
is is divisible by 2. so, i is not displyed
3
is is divisible by 2. so, i is not displyed
5
is is divisible by 2. so, i is not displyed
7
is is divisible by 2. so, i is not displyed
9
```

```
i=0
for i in range(1,10):
    if i%2==0 and i%4==0:
        print('is is divisible by 2 and 4. so, i is not displyed')
        continue
    print(i)
    i+=1
```

```
1
2
3
is is divisible by 2 and 4. so, i is not displyed
5
6
7
is is divisible by 2 and 4. so, i is not displyed
9
```

Pass

```
i=0
for i in range(1,10):
    if i%2!=0 and i%4!=0:
        print('i not is divisible by 2 and 4')
        pass
    else:
      print(i)
    i+=1
```

```
i not is divisible by 2 and 4
2
i not is divisible by 2 and 4
4
i not is divisible by 2 and 4
6
i not is divisible by 2 and 4
8
i not is divisible by 2 and 4
```

Pattern using for loop

```
for i in range(5):
    j=1
    for j in range(5):
        print('*', end=' ')
        j+=1
    i+=1
    print()
```

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
for i in range(5):
    j=1
    for j in range(5):
        if j<=i:
            print('*', end=' ')
        j+=1
    i+=1
    print()
```

```
*
* *
* * *
* * * *
* * * * *
```

Array

TypeCode	C Type	Python Type	Min. size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

```
from array import *
v=array('i',[1,2,3,4,-5,-6,8])
print(v.buffer_info())
```

```
(2793010740272, 7)
```

In the output, the first number is address of array. The 2nd number "7" is the size of the array

```
from array import *
v=array('i',[1,2,3,4,-5,-6,8])
print(v.typecode)
```

```
i
```

Type code =I, mean the array contains unsigned integer numbers. It can hold both negative and positive numbers

```
from array import *
v=array('i',[1,2,3,4,-5,-6,8])
v.reverse()
print(v)
```

```
array('i', [8, -6, -5, 4, 3, 2, 1])
```

```
from array import *
v=array('i',[1,2,3,4,-5,-6,8])
for i in range(len(v)):
    if i>5:
        print('the index is greater than 5')
        break
    print(v[i])
```

```
1
2
3
4
-5
-6
the index is greater than 5
```

```
from array import *
v=array('i',[])
ar_num=int(input('enter the length of array'))
for i in range(ar_num):
    x=int(input('enter index element'))
    v.append(x)
print(v)
```

```
enter the length of array5
enter index element2
enter index element3
enter index element6
enter index element9
enter index element8
array('i', [2, 3, 6, 9, 8])
```

This code is to enter array length and its index values from the user

```
from array import *
v=array('i',[34,23,43,12,34,56])
print(v)

vv=int(input('enter the element for search'))
i=0
for el in v:
    if el==vv:
        print(i)
        break
    i+=1
```

```
array('i', [34, 23, 43, 12, 34, 56])
enter the element for search23
1
```

This code is to display index number while giving

Insert Array from user input

```
m=int(input("Enter the value of m or Row\n"))
n=int(input("Enter the value of n or columns\n"))
out=[];
for i in range (m):
    r=[]
    for j in range(n):
        c=int(input(f"Enter the matrix [{i}][{j}]"))
        r.append(c)
    out.append(r)
for i in range(m):
    for j in range(i):
        print(out, end=' ')
```

```
Enter the value of m or Row
2
Enter the value of n or columns
2
Enter the matrix [0][0]34
Enter the matrix [0][1]12
Enter the matrix [1][0]56
Enter the matrix [1][1]78
[[34, 12], [56, 78]]
```

Numpy array

```
from numpy import *
ar=[1,34,45,65,67,89]
print(ar)
```

```
[1, 34, 45, 65, 67, 89]
```

In numpy array no need of specify the TypeCode

```
print(ar.dtype) int32
```

Creating array using functions

- array()
- linspace()
- logspace()
- arange()
- zeros()
- ones()

Linspace Used to create batch. If we do not specify the 3rd parameter in linspace, by default it will create 50 batch

```
lisp=linspace(1,10)
print(lisp)
```

```
[ 1.         1.18367347 1.36734694 1.55102041 1.73469388 1.91836735
  2.10204082 2.28571429 2.46938776 2.65306122 2.83673469 3.02040816
  3.20408163 3.3877551  3.57142857 3.75510204 3.93877551 4.12244898
  4.30612245 4.48979592 4.67346939 4.85714286 5.04081633 5.2244898
  5.40816327 5.59183673 5.7755102  5.95918367 6.14285714 6.32653061
  6.51020408 6.69387755 6.87755102 7.06122449 7.24489796 7.42857143
  7.6122449  7.79591837 7.97959184 8.16326531 8.34693878 8.53061224
  8.71428571 8.89795918 9.08163265 9.26530612 9.44897959 9.63265306
  9.81632653 10.        ]
```

```
ara=arange(1,10,2)
print(ara)
```

```
[1 3 5 7 9]
```

Arrange only used to display array value by a step of 3rd parameter.

```
log=logspace(1,10,5)
#print('%.2f'%log[2])
print(log)
```

The difference between two batch is log value of the 3rd parameter

```
[1.00000000e+01 1.77827941e+03 3.16227766e+05 5.62341325e+07
 1.00000000e+10]
```

```
z=zeros(5)
print(z)
```

```
[0. 0. 0. 0. 0.]
```

```
one=ones(5)
print(one)
```

```
[1. 1. 1. 1. 1.]
```

```
one=ones(5,int)
print(one)
```

```
[1 1 1 1 1]
```

Adding numpy array

```
ar1=array([1,2,3,4,5])
ar2=array([5,4,3,2,1])
add=ar1+ar2
print(add)
```

```
[6 6 6 6 6]
```

```
ar1=array([1,2,3,4,5])
ar2=array([5,4,3,2,1])
add=ar1+ar2
print(add)
sum=sum(add)
print(sum)
```

```
[6 6 6 6 6]
30
```

```
print(min(ar1))    1
print(max(ar2))    5
```

```
print(concatenate([ar1,ar2]))  [1 2 3 4 5 5 4 3 2 1]
```

Multidimensional array

```
from numpy import *
arr=array([[1,2,3],[4,3,4]])
print(arr.dtype)
print('====================')
print(arr.ndim)
print('====================')
print(arr.shape)
print('====================')
print(arr.size)
```

```
int32
====================
2
====================
(2, 3)
====================
6
```

```
ar2=arr.flatten()
print(ar2)          [1 2 3 4 3 4]
```

Changing MD array to single D

```
ar3=ar2.reshape(2,3)  [[1 2 3]
print(ar3)             [4 3 4]]
```

Changing SD array to MD array

```
ar1=array([[1,2,3,4,5,6],
           [4,3,4,5,7,8]])
arr1=ar1.reshape(2,2,3)
print(arr1)
```

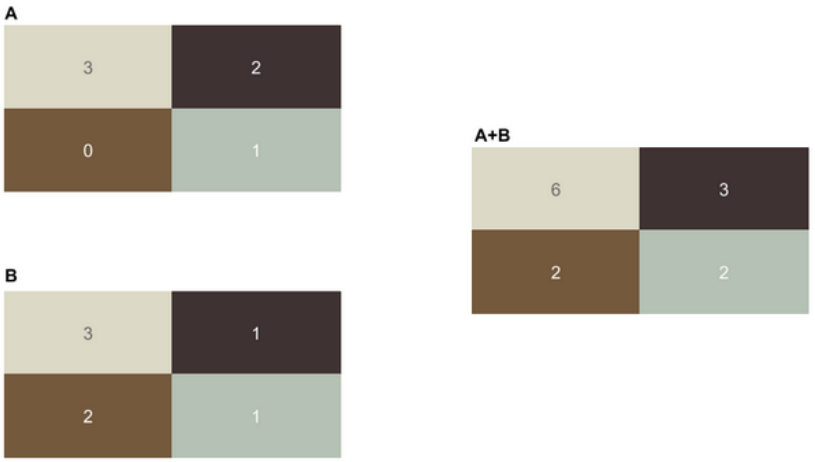
```
[[[1 2 3]
  [4 5 6]]

 [[4 3 4]
  [5 7 8]]]
```

Create 2, 2X3 array from 1, 2X6 array

Arithmetic operation on 2D array

Addition




```
print('Arithemic operation on numpy array')
ar1=array([[3,2],[0,1]])
ar2=array([[3,1],[2,1]])
add_ar=ar1+ar2
print(add_ar)
```

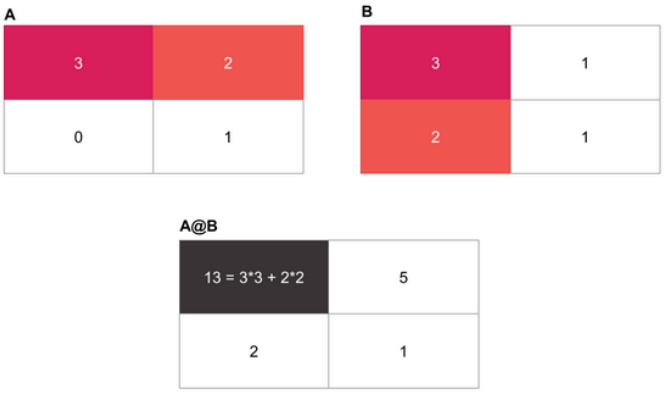
```
Arithemic operation on numpy array
[[6 3]
 [2 2]]
```

Matrix Multiplication

To perform a typical matrix multiplication (or matrix product), you can use the operator '@.'

This is how it works: the cell (1,1) (value: 13) in the output is a Sum-Product of Row 1 in matrix **ar1** (a two-dimensional array **ar1**) and Column 1 in matrix **ar2**.

Similarly, the cell (1,2) in the output is a Sum-Product of Row 1 in matrix **ar1** and Column 2 in matrix **ar2**.




```
mul=ar1@ar2    [[13  5]
print(mul)      [ 2  1]]
```

Array slicing

Python NumPy array slicing is used to extract some portion of data from the actual array. Slicing in python means extracting data from one given index to another given index, however, NumPy slicing is slightly different. Slicing can be done with the help of `(:)`. A NumPy array slicing object is constructed by giving `start`, `stop`, and `step` parameters to the built-in slicing function. This slicing object is passed to the array to extract some portion of the array.

The syntax of Python NumPy slicing is `[start : stop : step]`

`Start` : This index by default considers as '0'

`stop` : This index considers as a length of the array.

`step` : By default it is considered as '1'.

Using slicing operation we can extract elements of a 1-D NumPy array. For example,`arr[1:6]` syntax to slice elements from `index 1 to index 6` from the following 1-D array.

When we pass the elements from **index 3 to the end** as a parameter of the slicing operation, we will get elements of an array from index 3 to the last index.

In order to slicing of 1-D array pass elements from a **starting index to the 5th index** as a parameter of the slicing operation, we will get elements of an array from the start index to before mentioned ending index.

Use the step value to determine the step of the slicing. It returns every other element from the entire array. Follow the syntax of slicing [start: stop: step] here, parameters are separated by a colon (:).

```
print('Array slicing')
# Create NumPy arrays
arr = array([3, 5, 7, 9, 11, 15, 18, 22])
# Use slicing to get 1-D arrays elements
arr2 = arr[1:6]
print(arr2)
# Starting position from 3 to the end
arr2 = arr[3:]
print(arr2)
# Returns first 5 values (0 to 4 index)
arr2 = arr[:5]
print(arr2)
# Use step value to get elements
arr2 = arr[::3]
print(arr2)
```

```
Array slicing
[ 5  7  9 11 15]
[ 9 11 15 18 22]
[ 3  5  7  9 11]
[ 3  9 18]
```

Negative slicing

As a part of extracting elements of NumPy array from ending, we have to use negative slicing, which is extracting elements of an array from ending.

`arr[-4:-2]` to slice from index -4 to index -2 from the end. Minus operator to refer to an index from the end.

```
# Use negative slicing to get elements
arr2 = arr[-4:-2]
print(arr2)
```

```
[11 15]
```

Use Slicing With Interval 0f NumPy Arrays. `arr[3::4]` in this slice `3` is the starting point and `4` is the interval. So the returning array stars from the element in index three. After that, it takes every fourth element of the array until the end.

```
# Use slicing with interval
arr2 = arr[3::4]
print(arr2)
```

```
[ 9 22]
```

Slicing 2-Dimensional NumPy Arrays

Use slicing a 2-dimensional array in both axes to obtain a rectangular subset of the original array. You can use `arr[1:,1:3]` to select rows 1: one to the end of the bottom of the array and columns 1:3 (columns 1 and 2).

```
# Create NumPy arrays
arr = array([[3, 5, 7, 9, 11],
             [2, 4, 6, 8, 10]])
# Use slicing a 2-D arrays
arr2 = arr[1:, 1:3]
print(arr2)
```

```
[[4 6]]
```

Slicing 3-Dimensional NumPy Arrays

To extract elements of a 3-D NumPy array using slice operation first we have to create a 3-dimensional array and then, apply slice operation.

```
# Slicing 3-D arrays
arr = array([[[3, 5, 7, 9, 11],
              [2, 4, 6, 8, 10]],
             [[5, 7, 8, 9, 2],
              [7, 2, 3, 6, 7]]])
arr2 = arr[0, 1, 0:2]
print(arr2)
```

```
[2 4]
```

Functions

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses **(())**.
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```python
def myfirst():
    print('hello my first function in python')

i=0
for i in range(3):
    myfirst()
    i+=1
```

```
hello my first function in python
hello my first function in python
hello my first function in python
```

```python
def add(x,y):
    z = x + y
    print(z)

add(12,5)
```

```
17
```

```python
def add(x,y):
    z = x + y
    return z
re=add(12,5)
print(re)
```

```
17
```

```python
def add_and_substraction(x,y):
    z=x+y
    b=x-y
    return z,b
re1,re2=add_and_substraction(6,5)
print(re1,re2)
```

```
11 1
```

Even and odd number in function

```python
print('Even and odd number')
def even_odd(value):
    e=0
    od=0
    for i in value:
        if i%2==0:
            # return e
            e+=1
        else:
            # return od
            od+=1
    return e,od
evn,odd=even_odd([34,5,67,6,2,3,4,67,89,7])
print('even number',evn,end=' ')
print()
print('odd number',odd,end=' ')
```

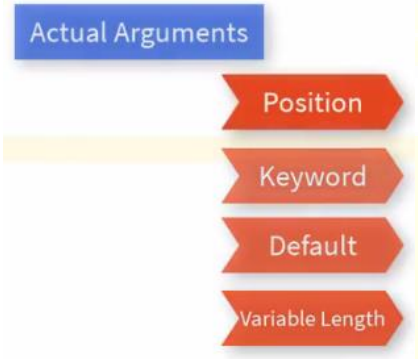
```
even number 4
odd number 6
```

Factorial

```
print('factorial')
def fact(x):
    f=1
    for i in range(1,x+1):
        f=f*i
    return f
result=fact(6)
print(result)
```

```
factorial
720
```

Function arguments




```
def fun(name,age):
    return name,age
res1,res2=fun('python',26)
print(res1,res2)
```

```
python 26
```

In **positional argument**, we have to pass the actual arguments in sequence of formal arguments

```
print('keyword arguments')
def fun(name,age):
    return name,age
res1,res2=fun(age=26,name='python')
print(res1,res2)
```

```
python 26
```

In **keyword argument**, we may not keep the sequence of formal argument while passing the actual argument. But we have to make assignment operation for each arguments

```
print('defualt arguments')
def fun(name,age=26):
    return name,age
res21,res23=fun('python')
print(res21,res23)
```

```
python 23
```

In **default argument**, we can assign the value in the formal argument. And we can leave giving actual argument while calling the function

```
print('Variable length arguments')
def var(x,*y):
    s=x
    for i in y:
        s=s+i
    return s
result=var(5,6,7,3,8)
print(result)
```

29

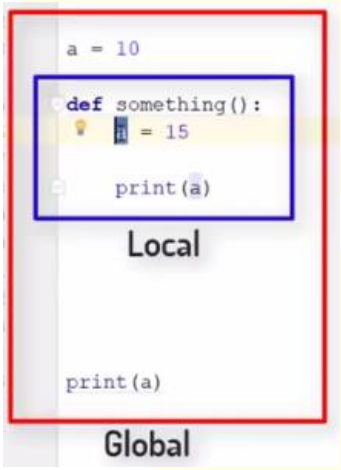
In **Variable Length argument**, we put **'*'** sign before the 2nd argument. This tells to the python compiler there will be more variable to passed as actual arguments.

Key worded variable length argument (**kwargs)

In ****kwargs**. In the formal argument we put **'**'** sign before the 2nd argument. This tells to the python compiler there will be more variable to passed as actual arguments that have different data types (the passed actual value is different item).

```
print('Key worded variable length arguments')
def kwvar(name,**var_length):
    print(name)
    for i,j in var_length.items():
        print(i,j)
kwvar(name='python',year=4,dep='ece',fac='tech',sem='1')
```

```
Key worded variable length arguments
python
year 4
dep ece
fac tech
sem 1
```




```
print('Local and global Variable')
x=34
def func():
    x=3
    print('X in side function',x)
func()
print('X global',x)
```

```
Local and global Variable
X in side function 3
X global 34
```

To access the variable value outside of the function, we have to define the variable inside the function as **global 'variable name'**

```
print('Recursive function')
import sys
sys.setrecursionlimit(8)
print(sys.getrecursionlimit())
i=0
def rec():
    global i
    i += 1
    print('hello Python',i)
    rec()
rec()
```

```
Recursive function
8
hello Python 1
hello Python 2
hello Python 3
hello Python 4
```

We call this code **recursive function**

```
print('Factorial using Recursive function')
def re_fact(x):
    if x==0:
        return 1
    return x*re_fact(x-1)
res=re_fact(6)
print(res)
```

Factorial using **recursive function**

```
Factorial using Recursive function
720
```

Modularization

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b
```

Mudul_calc.p

```
print('Module calling')
import module_calc
ad=module_calc.add(10,5)
sub=module_calc.sub(10,5)
mul=module_calc.mul(10,5)
div=module_calc.div(10,5)
print(ad,sub,mul,div)
```

```
Module calling
15 5 50 2.0
```

Importing **module_calc.py** in **calling_module.py**

Object oriented program in python

Object-oriented programming (OOP) refers to the programming language in which the coders/developers explicitly define the data types, data structures, and also the types of functions that can be applied to the data structures. Thus, the data structures become “objects” incorporating both data and functions.

An object has two characteristics:

- attributes
- behavior

1. The self

1. Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it
2. If we have a method that takes no arguments, then we still have to have one argument.
3. This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.

2. The __init__ method

The __init__ method is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
class python:
    def method(self):
        print('val_1','val_2')
obj_1=python()
obj_2=python()
python.method(obj_1)
python.method(obj_2)
```

```
val_1 val_2
val_1 val_2
```

```python
print('Sele and __init__')
class python:
    # class attribute
    attr1 = "python class"
    # Instance attribute, used as a construcor
    def __init__(self, name):
        self.name = name
    def method(self):
        print("My name is {}".format(self.name))
# Driver code
# Object instantiation
batch = python("4th")
cl = python("class")
# Accessing class methods
batch.method()
cl.method()
```

```
Sele and __init__
My name is 4th
My name is class
```

Inheritance in python

```python
# parent class
class Person(object):
    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber
    def display(self):
        print(self.name)
        print(self.idnumber)
    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post
        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)
    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))
# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")
# calling a function of the class Person using its instance
a.display()
a.details()
```

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

Python exercises

1. Write a Python program to get the Identity, Type, and Value of an object
2. Write a Python program to print the following string in a specific format (see the output)
   Sample String : "Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Like a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are" *Output :*

```
Twinkle, twinkle, little star,
        How I wonder what you are!
                Up above the world so high,
                Like a diamond in the sky.
Twinkle, twinkle, little star,
        How I wonder what you are
```

2. Write a Python program which accepts the radius of a circle from the user and compute the area.
   Sample Output :
   r = 1.1
   Area = 3.8013271108436504
3. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them
4. Write a Python program which accepts a sequence of comma-separated numbers from user and generate a list and a tuple with those numbers.
   Sample data : 3, 5, 7, 23
   Output :
   List : ['3', ' 5', ' 7', ' 23']
   Tuple : ('3', ' 5', ' 7', ' 23')
5. Write a Python program to display the first and last colors from the following list.
   color_list = ["Red","Green","White" ,"Black"]
6. Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn. *Sample value of n is* 5
   Expected Result : 615
7. Write a Python program to get the volume of a sphere with radius 6.
8. Write a Python program to get the difference between a given number and 17, if the number is greater than 17 return double the absolute difference. Hint you can use Function.
9. Write a Python program to test whether a number is within 100 of 1000 or 2000

10. Write a Python program to calculate the sum of three given numbers, if the values are equal then return three times of their sum.
11. Write a Python program to find whether a given number (accept from the user) is even or odd, print out an appropriate message to the user.
12. Write a Python program to count the number 4 in a given list.
13. Write a Python program to print all even numbers from a given numbers list in the same order and stop the printing if any numbers that come after 237 in the sequence.
    Sample numbers list :

    numbers = [386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328, 615, 953, 399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217, 815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445, 742, 717, 958,743, 527]
14. Write a Python program to print out a set containing all the colors from color_list_1 which are not present in color_list_2
    Test Data :
    color_list_1 = set(["White", "Black", "Red"])
    color_list_2 = set(["Red", "Green"])
    Expected Output :
    {'Black', 'White'}
15. Write a Python program that will accept the base and height of a triangle and compute the area
16. Write a Python program to sum of two given integers. However, if the sum is between 15 to 20 it will return 20
17. Write a Python program to display your details like name, age, address in three different lines
18. Write a Python program to parse a string to Float or Integer.
19. Write a Python program to calculate body mass index
20. Write a Python program to calculate sum of digits of a number.
21. Write a Python program to concatenate N strings.
22. Write a Python program to count the number occurrence of a specific character in a string

23. Write a Python function that takes a positive integer and returns the sum of the cube of all the positive integers smaller than the specified number.
24. Write a Python function to find the maximum and minimum numbers from a sequence of numbers
25. Write a Python function to check whether a number is divisible by another number. Accept two integers' values form the user.
26. Write a python program to convert decimal to hexadecimal.
    Sample decimal number: 30, 4
    Expected output: 1e, 04
27. Write a Python program to convert an integer to binary keep leading zeros. Sample data : x=12
    Expected output : 00001100
    0000001100
28. Write a Python program to input two integers in a single line.
29. Write a Python program to check whether lowercase letters exist in a string.
30. Write a Python program using function to determine the largest and smallest integers, longs, floats.
31. Write a Python program to calculate the Fibonacci sequence of 5