PYTHON BASICS

Nyeste Venture

Introduction
To Python

Why Python

- Python works on different platforms (Windows, Linux, Mac, Raspberry Pi , ect).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written.
- Python can be treated in a an object-oriented way.

Python works on different platforms

- **Windows :** used on various versions- Windows os, including Windows 10, Windows 8, and Windows 7.
- **Linux**: supported on various Linux distributions, such as Ubuntu, Fedora, CentOS, and more.
- **Web Development**: used to build web applications using frameworks like Django, Flask, and Pyramid.
- **Data Science and Machine Learning**: used in with libraries like NumPy, pandas, scikit-learn, TensorFlow, and PyTorch.

- **Embedded Systems**: used for programming embedded systems and microcontrollers, with platforms like Raspberry Pi, Arduino, and MicroPython.
- **Game Development**: used for game development using libraries like Pygame and Panda3D.
- **Desktop Applications**: used to create cross-platform desktop applications using libraries like PyQt, wxPython, and Tkinter.
- **Scientific Computing**: used in scientific computing and simulations, often alongside libraries like SciPy and SymPy.

Founder of python language

Guido van Rossum

High level language

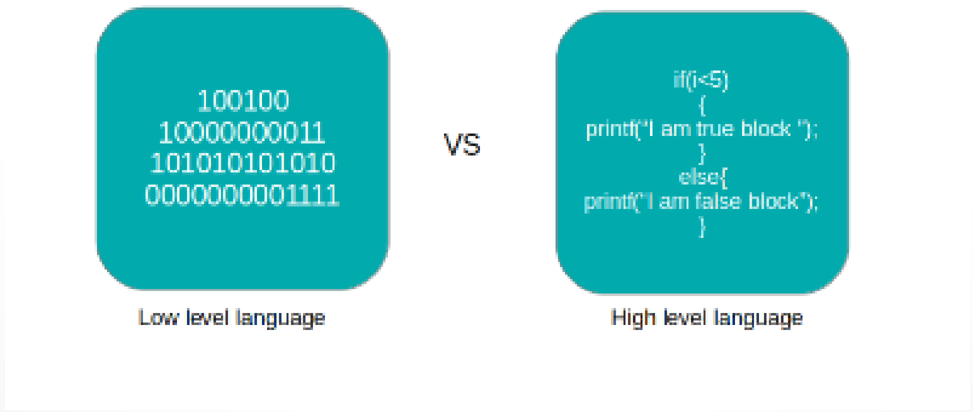
- A programmer can easily understand
- *Translator*

  Compiler/Interpreter

Low level language

- A machine can easily understand it
- *Translator*

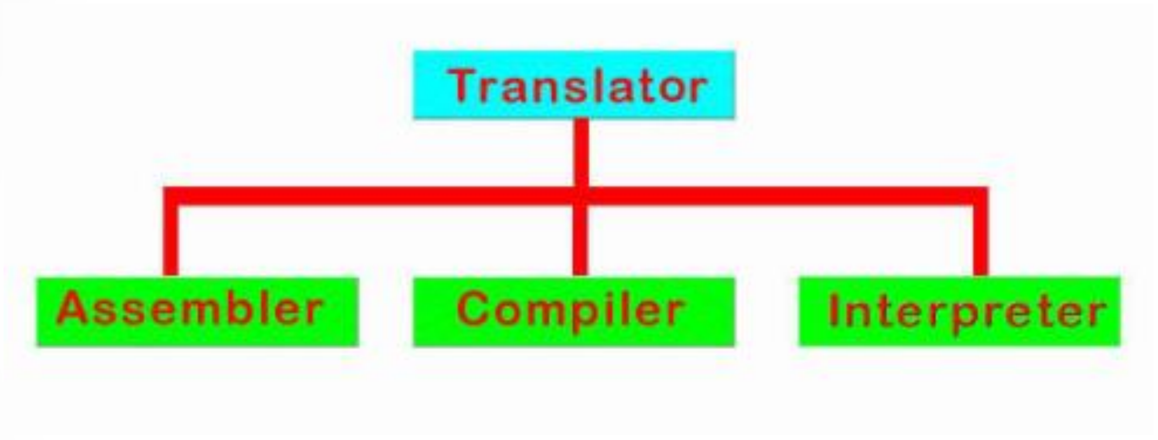
  Assembler



```
100100
10000000011
101010101010
0000000001111
```

```
if(i<5)
{
printf("I am true block ");
}
else{
printf("I am false block");
}
```

3 type of language translators

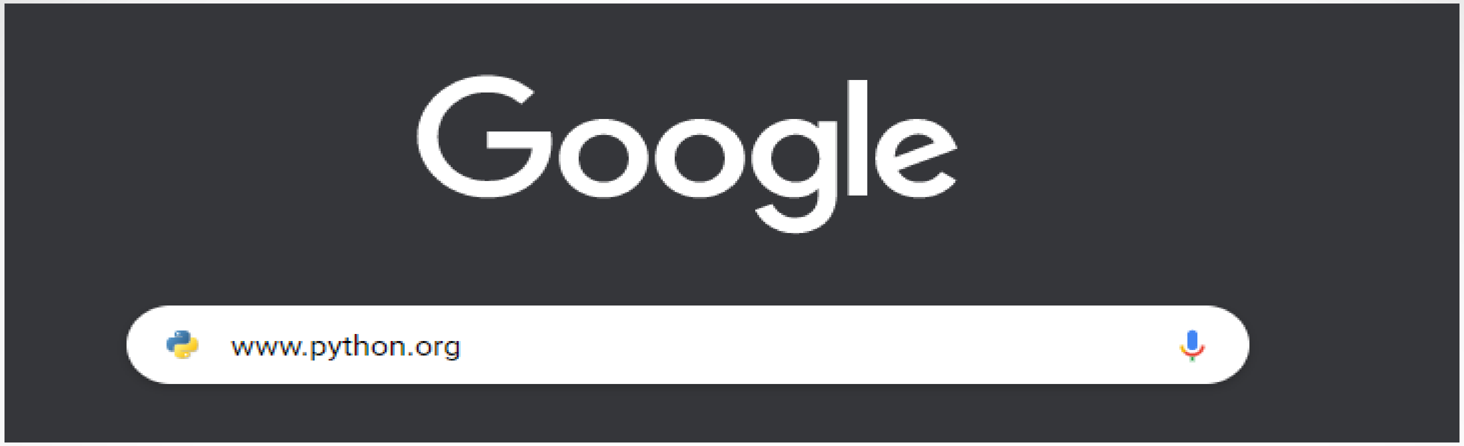


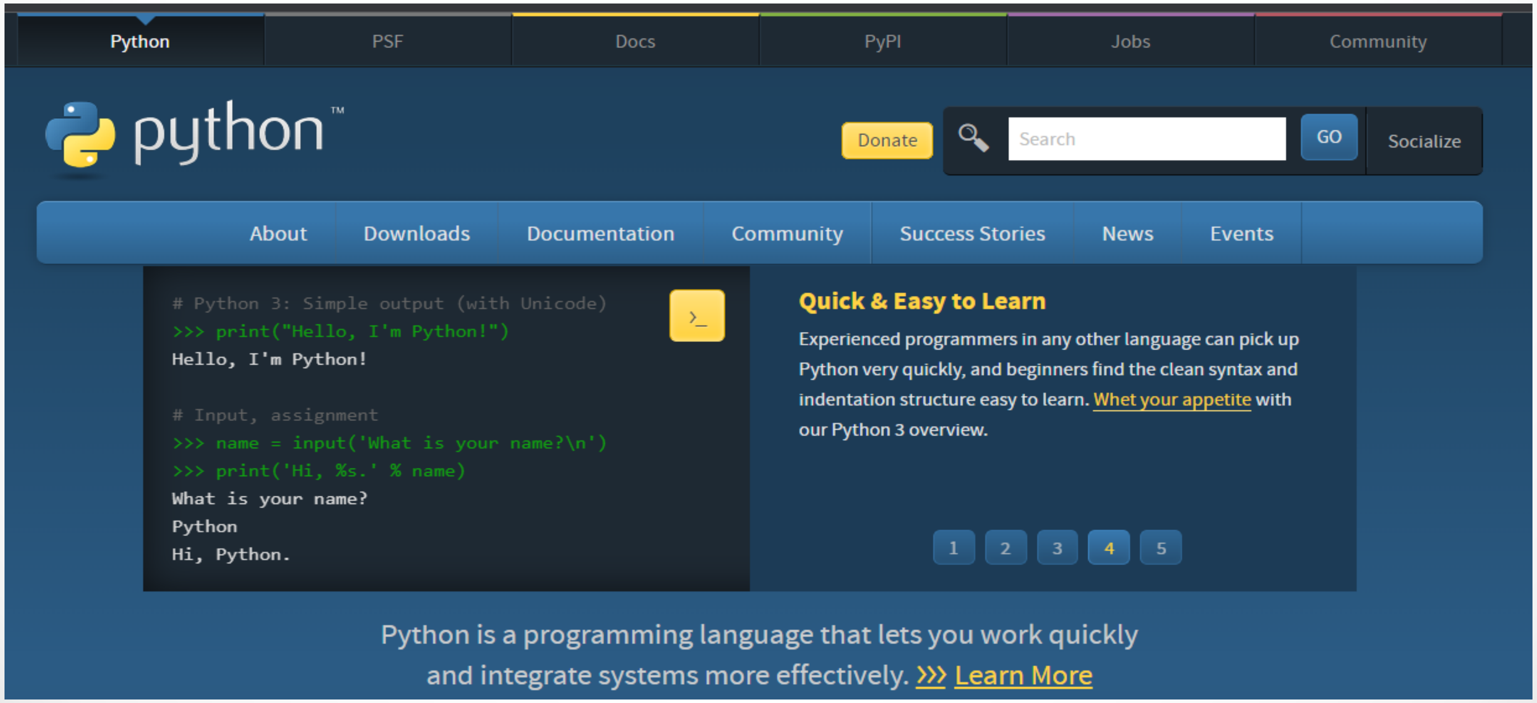
Difference Between Compiler Interpreter and Assembler

- **Compiler**
  - ➢ Is a software that converts a high level language program into machine language.
  - ➢ A compiler translates the entire source code in a single run
- **Interpreter**
  - ➢ Is a software that converts a high level language program into machine language.
  - ➢ An interpreter translates the entire source code line by line
- **Assembler**
  - ➢ Is a software that converts assembly language program into machine language.
  - ➢ Low level into low level

Python installation

- Go to python official site (https://www.python.org/)
- Download the latest version (3.10.1-windows 7 and above,bit-64).
- **Editor:**
- Notepad, Notepad++, Sublime, Atom, Visual Studio Code
- **Ide:**
- Python IDLE , Pycharm



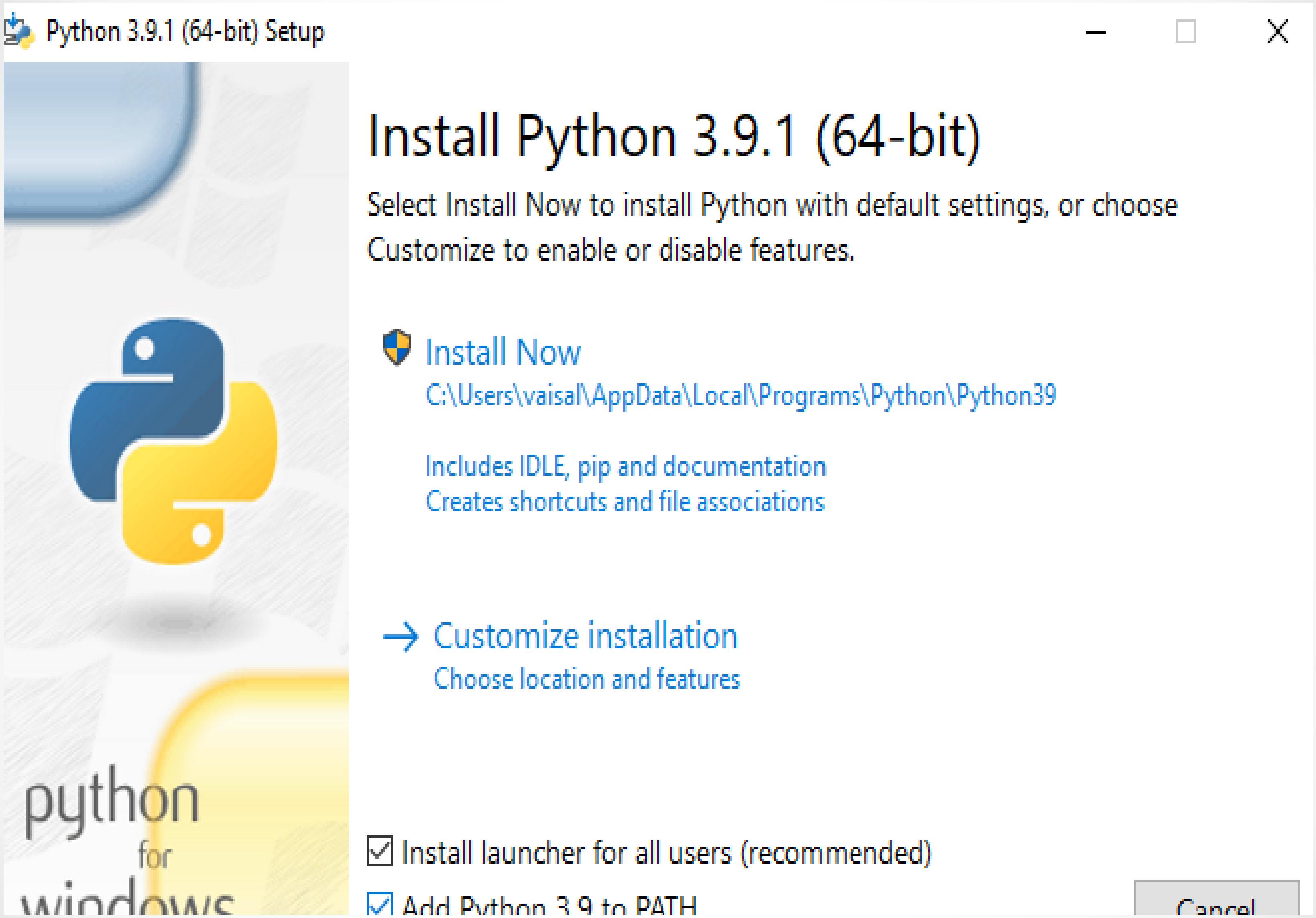


```
# Python 3: Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!

# Input, assignment
>>> name = input('What is your name?\n')
>>> print('Hi, %s.' % name)
What is your name?
Python
Hi, Python.
```

Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More



Python 3.9.1 (64-bit) Setup

Install Python 3.9.1 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

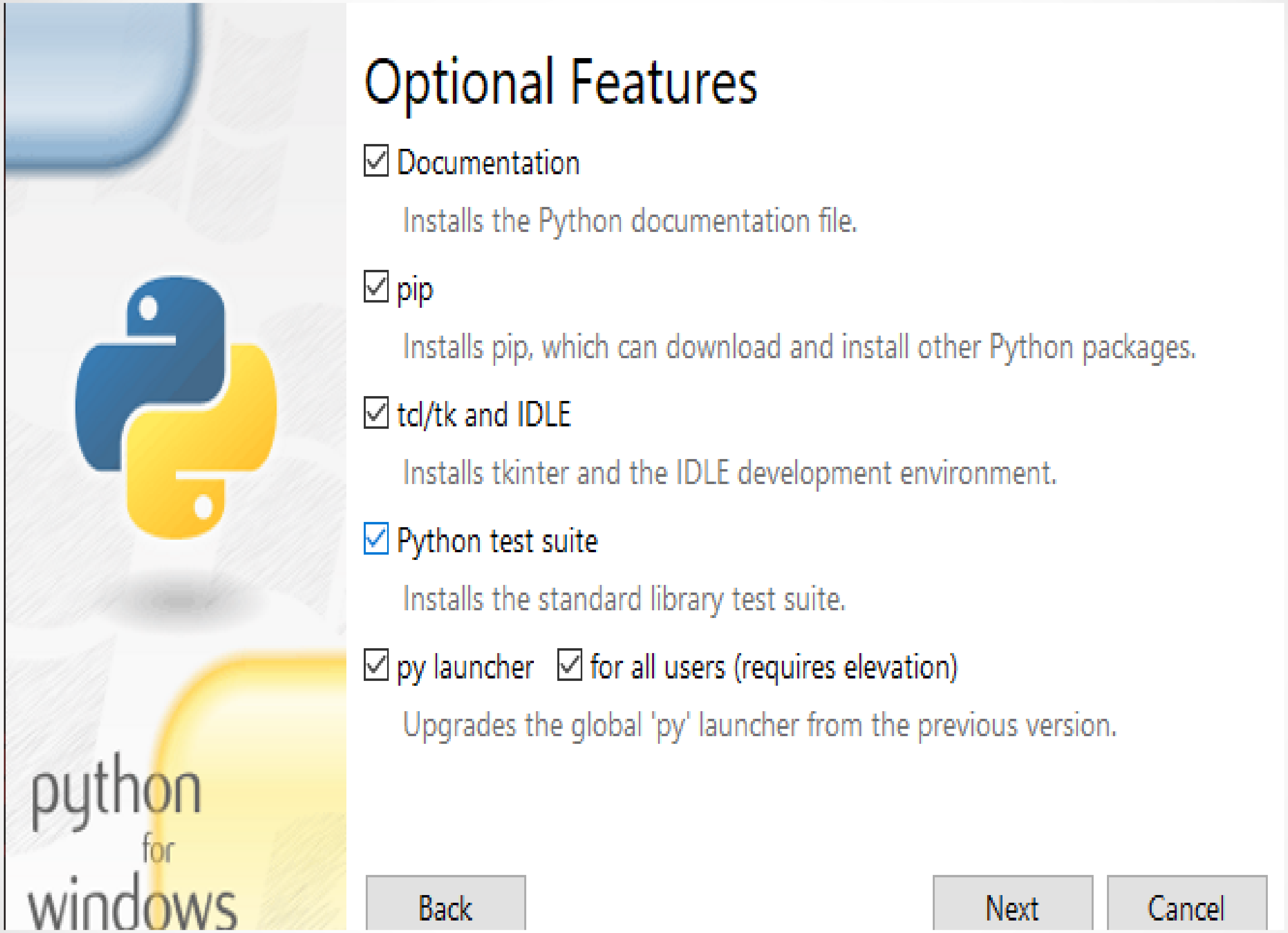
Install Now
C:\Users\vaisal\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)

☑ Add Python 3.9 to PATH


Optional Features

☑ Documentation

Installs the Python documentation file.

☑ pip

Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE

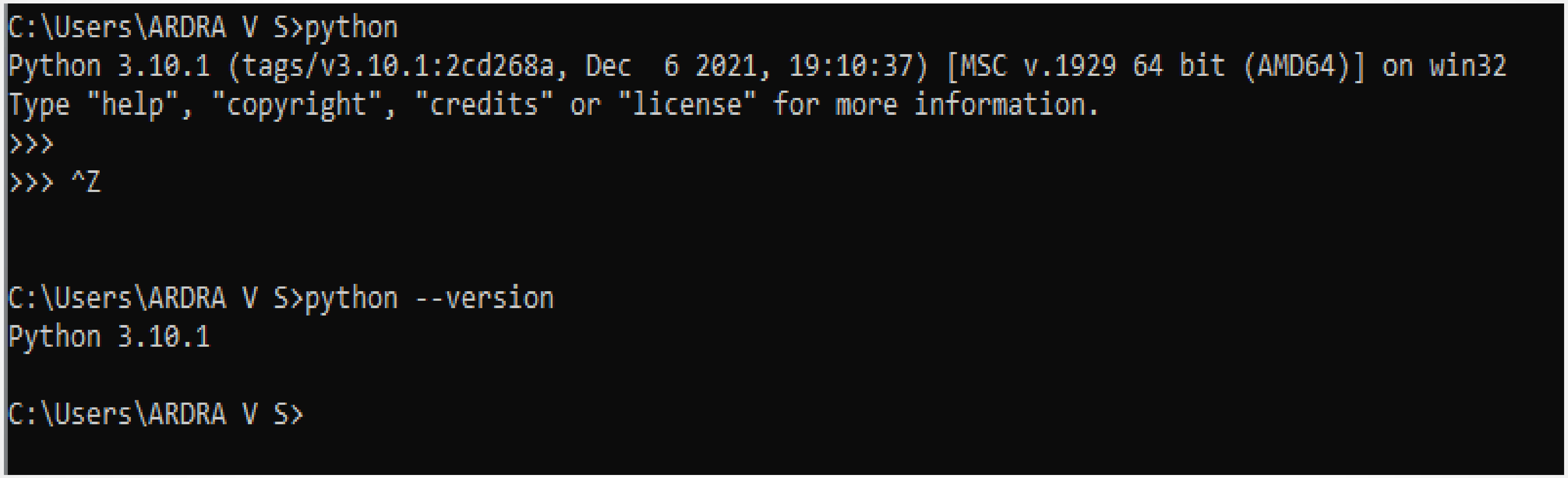
Installs tkinter and the IDLE development environment.

☑ Python test suite

Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)

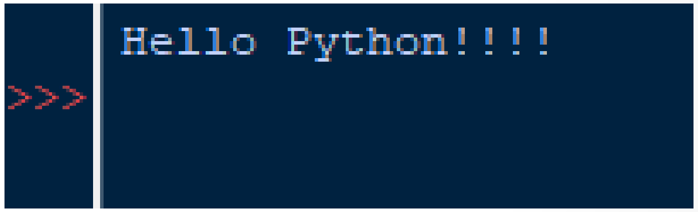
Upgrades the global 'py' launcher from the previous version.



```
C:\Users\ARDRA V S>python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> ^Z


C:\Users\ARDRA V S>python --version
Python 3.10.1

C:\Users\ARDRA V S>
```

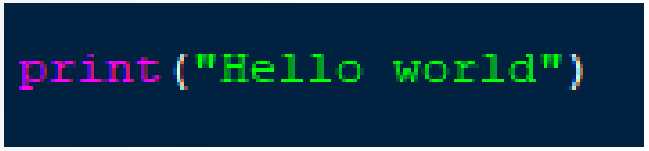
First Python program

1) Open python IDLE
2) File → new file
3) print("Hello Python!!!!")
4) Save as → Save.py
5) Run



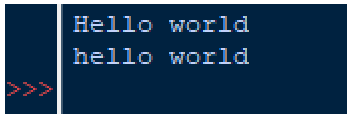
```
Hello Python!!!!
>>>
```

Q.2

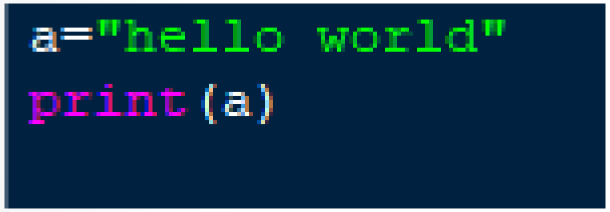


```python
print("Hello world")
```

Output



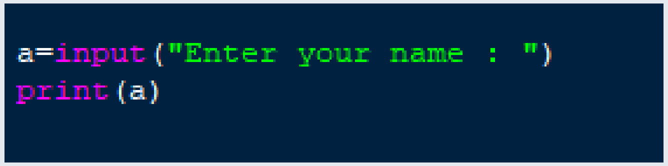
```
Hello world
hello world
>>>
```



```python
a="hello world"
print(a)
```

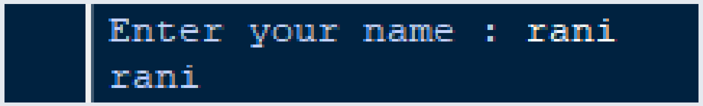
Input data

For string



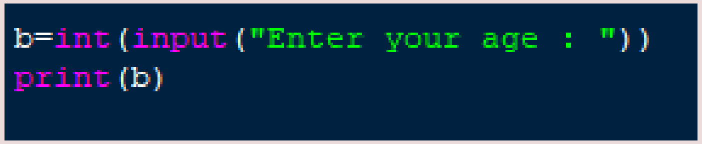
```
a=input("Enter your name : ")
print(a)
```

- output



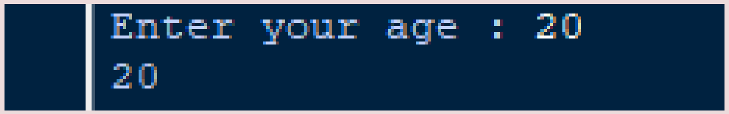
```
Enter your name : rani
rani
```

For int



```
b=int(input("Enter your age : "))
print(b)
```

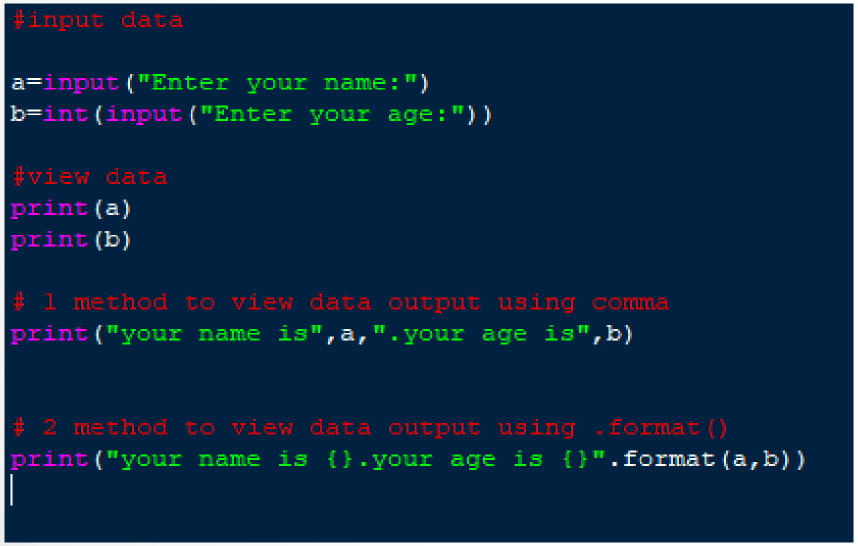
- output



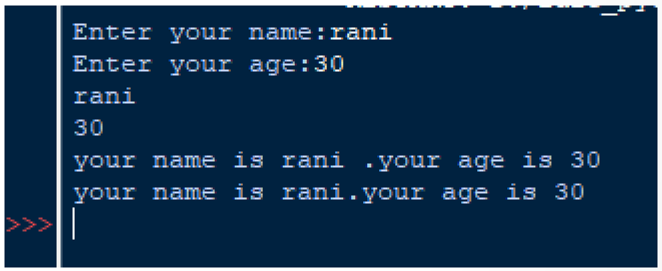
```
Enter your age : 20
20
```

Input and view data

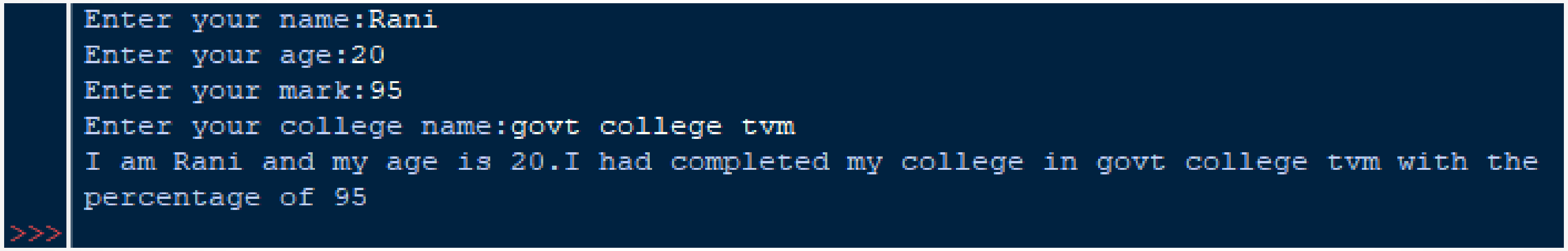
View the output using comma(,) & .format() method



```
#input data

a=input("Enter your name:")
b=int(input("Enter your age:"))

#view data
print(a)
print(b)

# 1 method to view data output using comma
print("your name is",a,".your age is",b)


# 2 method to view data output using .format()
print("your name is {}.your age is {}".format(a,b))
```



```
Enter your name:rani
Enter your age:30
rani
30
your name is rani .your age is 30
your name is rani.your age is 30
>>>
```



```
Enter your name:Rani
Enter your age:20
Enter your mark:95
Enter your college name:govt college tvm
I am Rani and my age is 20.I had completed my college in govt college tvm with the
percentage of 95
>>>
```

```
x=input("Enter your name:")
y=int(input("Enter your age:"))
z=int(input("Enter your mark:"))
s=input("Enter your college name:")

print("I am ",x,"and my age is ",y,".I had completed my college in ",s," with the percentage of ",z)
```

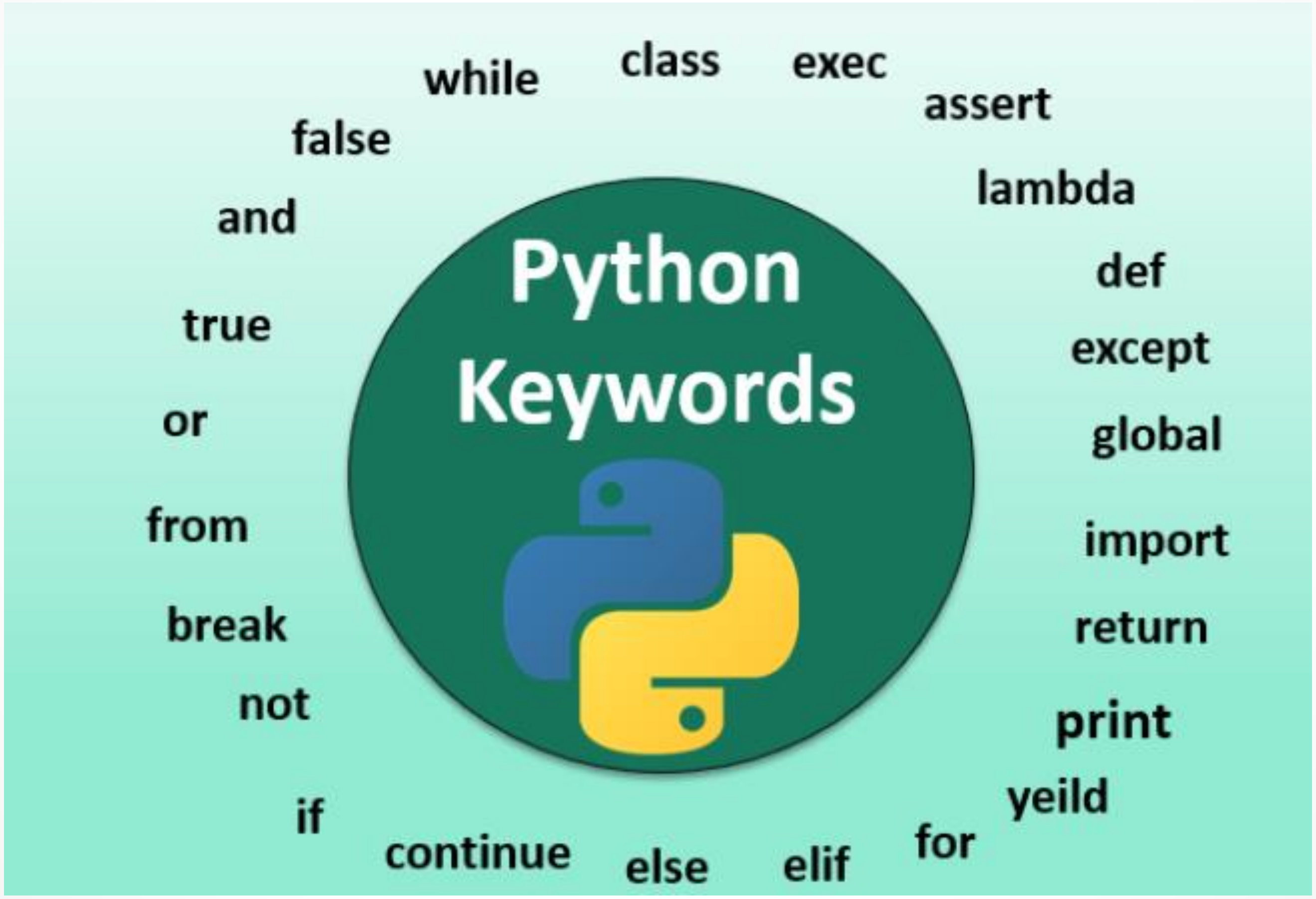
```
a=input("Enter your name:")
b=int(input("Enter your age:"))
c=int(input("Enter your mark:"))
d=input("Enter your college name:")

print("I am {} and my age is {}.I had completed my college in {} with the percentage of {}".format(a,b,d,c))
```

Tokens

- A token is the smallest individual unit in a python program.
- All statements and instructions in a program are built with tokens.
- Keyword
- Identifier
- Literal
- Operator

KEYWORDS

- Keywords are the **reserved words** in Python.
- We cannot use a keyword as a **variable name** , **function name** or any other **identifier.**
- They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive.


while
class
exec
assert
false
lambda
and
Python Keywords
def
true
except
or
global
from
import
break
return
not
print
yeild
if
continue
else
elif
for

Identifier

- Identifiers are names that you give to a **variable**, **Class**, or **Function.**
- There are certain rules for naming identifiers similar to the variable declaration rules,
- such as:

  An Identifier starts with a letter (A to Z or a to z), or an underscore(_) followed by zero or more letters or under-scores and digits(0 to 9)
- It is Case sensitive.

Example for identifiers:

variable_name = 123

class **className**:

____

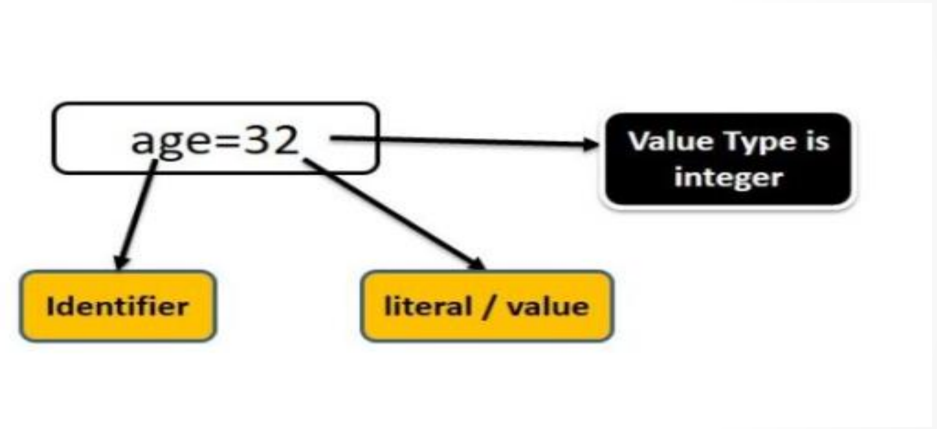
def **funName()**:

____

Literal

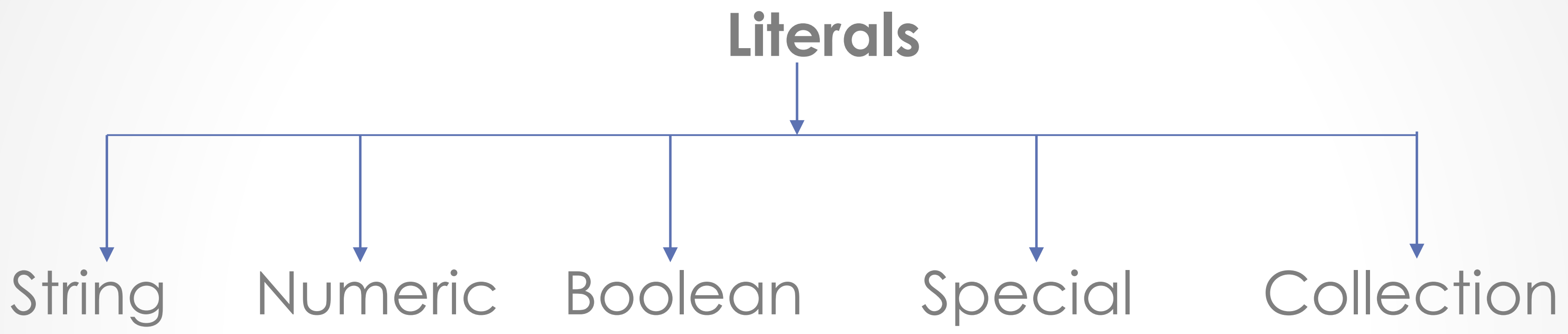
Python Literals can be defined as data that is given in a variable or constant

- **String literals**
- **Numeric literals**
- **Boolean literals**
- **Literal Collections**
- **Special literals**



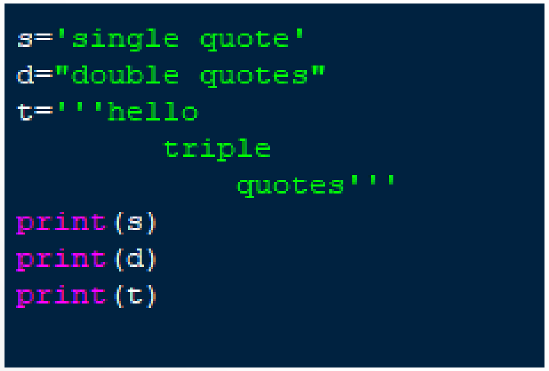
age is variable

32 is literal

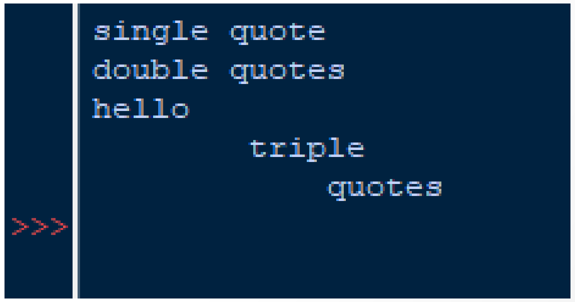

Literals
String
Numeric
Boolean
Special
Collection

String Literal

- A string literal can be created by writing a text (a group of Characters) surrounded by the
  - ➢ single('')
  - ➢ double("")
  - ➢ triple quotes(''' ''').
- By using triple quotes we can write multi-line strings.



```
s='single quote'
d="double quotes"
t='''hello
        triple
            quotes'''
print(s)
print(d)
print(t)
```



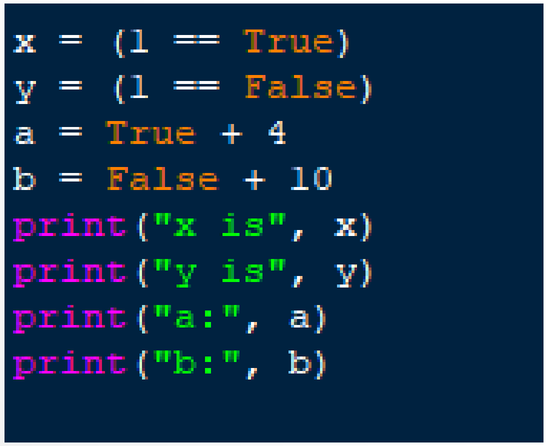
```
single quote
double quotes
hello
        triple
            quotes
>>>
```

Numeric Literal

- Numeric Literals are mutable (changeable).
- Numeric literals can belong to 3 different numerical types
  - **Integer**
  - **Float**
  - **Complex**

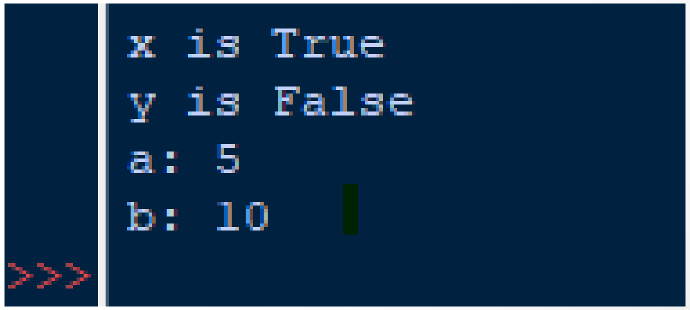
Boolean Literal

- A Boolean literal can have any of the two values
  - **True**
  - **False**



```
x = (1 == True)
y = (1 == False)
a = True + 4
b = False + 10
print("x is", x)
print("y is", y)
print("a:", a)
print("b:", b)
```

Output:



```
x is True
y is False
a: 5
b: 10
>>>
```

- In the above program, we use Boolean literal **True** and **False**
- In python,
  - **True** represents the value as **1**
  - **False** as **0**.
- The value of **x** is **True** because **1** is equal to **True**.
- The value of **y** is **False** because **1** is not equal to **False**.
- Similaraly,
- we can use the **True** and **False** in numeric expressions as the value.
- The value of **a** is **5** because we add **True** which has a value of **1** with **4**.
- The value of **b** is **10** because we add the **False** having value of **0** with **10.**

Special Literals

- Python contains one special literal ie, **None**.
- We use it to specify that the field has not been created

```
drink = "Available“
 food = None
 def menu(x):
                if x == drink:
                                print(drink)
                else:
                                print(food)
menu(drink)
menu(food)
```

output

```
Available
None
```

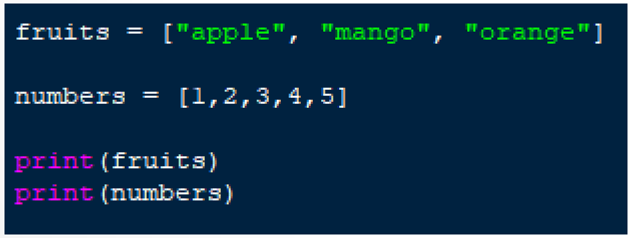
Literal Collection

- There are four different literal collections

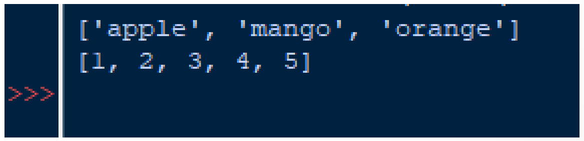
  - **List Literals - [ ]**
  - **Tuple Literals - ( )**
  - **Dictionary Literals - {‘ key ’ : value}**
  - **Set Literals - { }**

List Literal

- List contains items of different data types.
- Lists are mutable i.e., modifiable.
- The values stored in List are separated by comma(,) and enclosed within square brackets([]).
- We can store different types of data in a List.



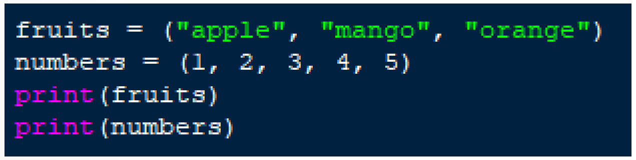
```python
fruits = ["apple", "mango", "orange"]

numbers = [1,2,3,4,5]

print(fruits)
print(numbers)
```



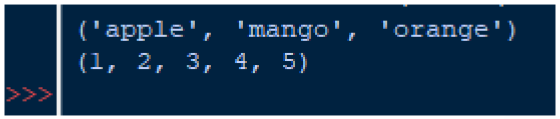
```
['apple', 'mango', 'orange']
[1, 2, 3, 4, 5]
>>>
```

Tuple Literal

- Python tuple is a collection of different data-type.
- It is immutable which means it cannot be modified after creation.
- It is enclosed by the parentheses () and each element is separated by the comma(,).



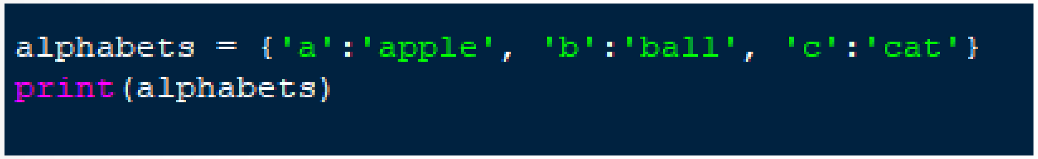
```
fruits = ("apple", "mango", "orange")
numbers = (1, 2, 3, 4, 5)
print(fruits)
print(numbers)
```



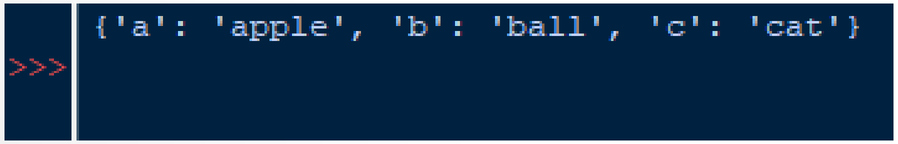
```
('apple', 'mango', 'orange')
(1, 2, 3, 4, 5)
>>>
```

Dictionary Literals

- Python dictionary stores the data in the key-value pair.
- It is enclosed by curly-braces {} and each pair is separated by the commas(,).



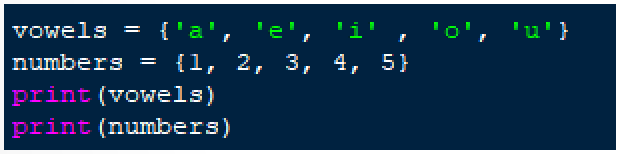
```
alphabets = {'a':'apple', 'b':'ball', 'c':'cat'}
print(alphabets)
```



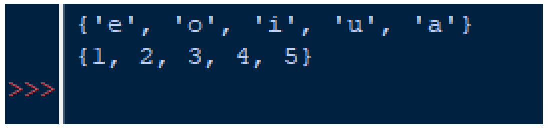
```
{'a': 'apple', 'b': 'ball', 'c': 'cat'}
>>>
```

Set Literals

- Python set is the collection of the unordered dataset.
- It is enclosed by the {} and each element is separated by the comma(,).



```
vowels = {'a', 'e', 'i' , 'o', 'u'}
numbers = {1, 2, 3, 4, 5}
print(vowels)
print(numbers)
```



```
{'e', 'o', 'i', 'u', 'a'}
{1, 2, 3, 4, 5}
>>>
```

OPERATORS

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic operators

- Arithmetic operators are used with numeric values to perform common mathematical operations:

OPERATOR	NAME	EXAMPLE
+	Addition	X + Y
-	Subtraction	X -Y
*	Multiplication	X * Y
/	Division	X / Y
%	Modulus	X % Y
**	Exponentiation	X ** Y
//	Floor division	X // Y

Assignment operators

Assignment operators are used to assign values to variables:

OPERATOR	EXAMPLE	SAME AS
=	X = 3	X = 3
+=	X += 3	X = X + 3
-=	X -= 3	X = X – 3
/=	X /= 3	X = X / 3
%=	X %= 3	X = X % 3
//=	X //= 3	X = X // 3
**=	X **= 3	X = X ** 3
&=	X &= 3	X = X & 3
|=	X |= 3	X = X | 3
^=	X ^= 3	X = X ^ 3
>>=	X >>= 3	X = X >> 3
<<=	X <<= 3	X = X << 3

Comparison operators

- Comparison operators are used to compare two values:

OPERATORS	NAME	EXAMPLE
==	Equal	X == Y
!=	Not Equal	X != Y
>	Greater than	X > Y
<	Less than	X < Y
>=	Greater than or equal to	X >= Y
<=	Less than or equal to	X <= Y

Logical operators

Logical operators are used to combine conditional statements:

OPERATOR	DESCRIPTION	EXAMPLE
and	Returns true If both statements are true	x < 5 and x <10
or	Returns true If one of the statements is true	x <5 or x<4

not	Reverse the result
	Returns false if the result is true
		not(x <5 and x < 10)
		not(x <5 or x < 4)

Identity operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:
- In Python, identity operators are used to compare the memory addresses of two objects to determine whether they refer to the same object in memory. Identity operators are used to test the identity, or sameness, of two objects, rather than just comparing their values.

OPERATORS	DESCRIPTION	EXAMPLE
is	Returns true If both variables are the same object in memory .	x is y
is not	Returns true If both variables are not the same object in memory	x is not y

- x = [1, 2, 3]
- y = [1, 2, 3]
- z = x
- print(x is y) # False, x and y are different objects with the same values
- print(x is z) # True, x and z refer to the same object in memory
- print(x is not y) # True, x and y are not the same object
- print(x is not z) # False, x and z are the same object

Membership operators

Membership operators are used to test if a sequence is presented in an object:

Membership operators in Python are used to test whether a value is a member of a sequence (such as a string, list, tuple, or set) or not.

These operators return a Boolean value (**True** or **False**) based on whether the specified value is found in the given sequence.

OPERATORS	DESCRIPTION	EXAMPLE
in	Returns true If a sequence with the specified value is present in the object	x in y
not in	Returns true If a sequence with the specified value is not present in the object	x not in y

- fruits = ['apple', 'banana', 'cherry', 'grape', 'orange']
- print('banana' in fruits)

  # True, 'banana' is present in the list
- print('kiwi' in fruits)

  # False, 'kiwi' is not present in the list
- print('kiwi' not in fruits)

  # True, 'kiwi' is not present in the list

Bitwise operators

- Bitwise operators are used to compare (binary) numbers:

OPERATOR	NAME	DESCRIPTION
&	AND	Set each bit to 1 If both bits are 1
|	OR	Set each bit to 1 If one of two bits is 1
^	XOR	Set each bit to 1 If only one of two bits is 1
~	NOT	Inverts all the bits

Variables

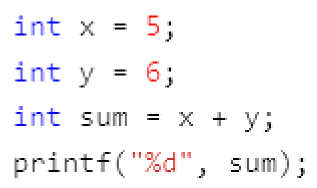
- Creating Variables
- Casting & type
- Rules
- One Value To Multiple Variables
- Variable types
  1. Global variable
  2. Local variables

Creating Variables

- Python has no command for declaring a variable in python .
- A variable is created the moment you first assign a value to it.
- Variables are case-sensitive
- EXAMPLE :
  - ➢ x = 5
  - ➢ Y = “Hello”

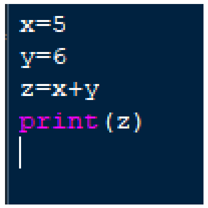
- Variables do not need to be declared with any particular *type in python language*.

- In c



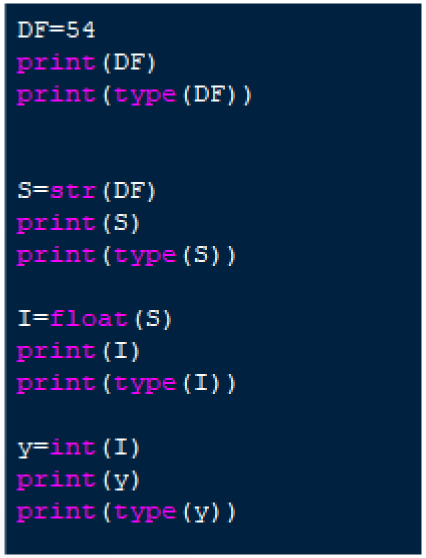
```
int x = 5;
int y = 6;
int sum = x + y;
printf("%d", sum);
```

- In python

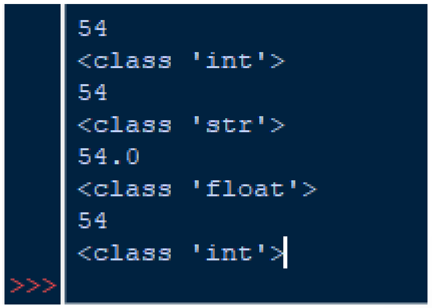


```
x=5
y=6
z=x+y
print(z)
```

- Variables can even change type after they have been set.



```
DF=54
print(DF)
print(type(DF))


S=str(DF)
print(S)
print(type(S))

I=float(S)
print(I)
print(type(I))

y=int(I)
print(y)
print(type(y))
```



```
54
<class 'int'>
54
<class 'str'>
54.0
<class 'float'>
54
<class 'int'>
>>>
```

Casting & Type

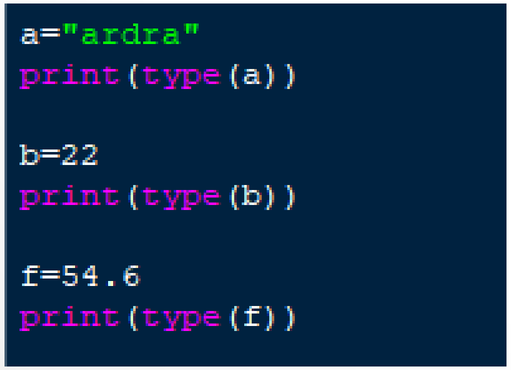
- **Casting**
- If you want to specify the data type of a variable, this can be done with casting.

```
 str(3      # x will be '3'
yint(3      # y will be 3
 float 3    # z will be 3.0
```

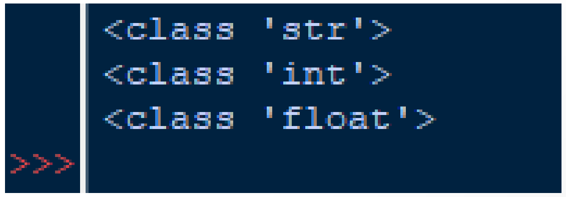
- **Get the Type**

You can get the data type of a variable with the type() function

type() is a **built in function in python** .



```
a="ardra"
print(type(a))

b=22
print(type(b))

f=54.6
print(type(f))
```

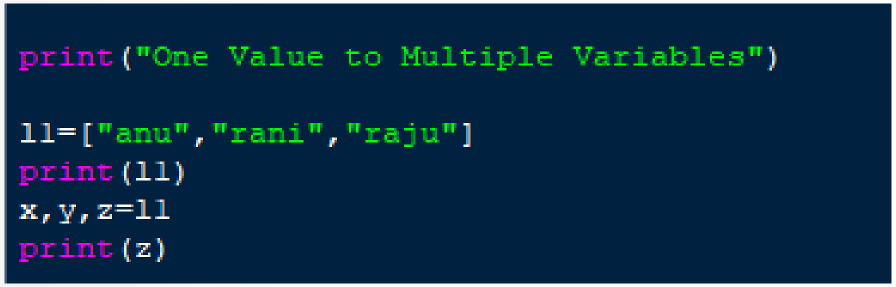


```
<class 'str'>
<class 'int'>
<class 'float'>
>>>
```

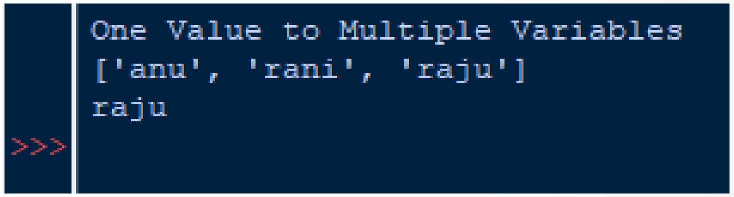
Variable Names (Rules)

- A variable can have a
  - ➢ short name (like x and y)
  - ➢ more descriptive name (age, car_name, total_volume).
- **Rules for Python variables :**
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Symbols can included in middle.
- Variable names are case-sensitive (age, Age and AGE are three different variables)

One Value to Multiple Variables



```
print("One Value to Multiple Variables")

l1=["anu","rani","raju"]
print(l1)
x,y,z=l1
print(z)
```

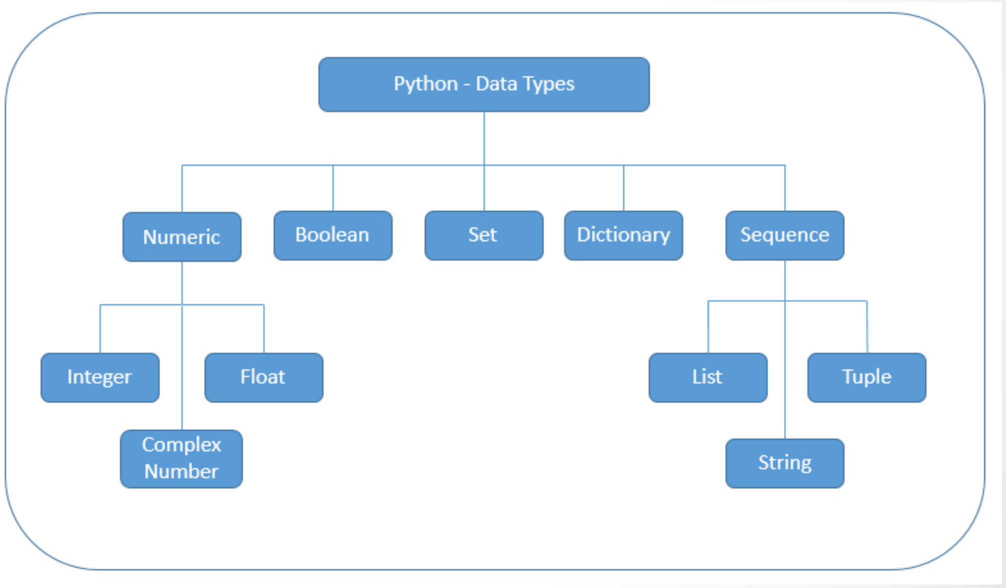


```
One Value to Multiple Variables
['anu', 'rani', 'raju']
raju
>>>
```

Python Data Types

- Numeric Types:
  - complex,
  - float,
  - int
- Text Type: str
- Boolean Type: bool

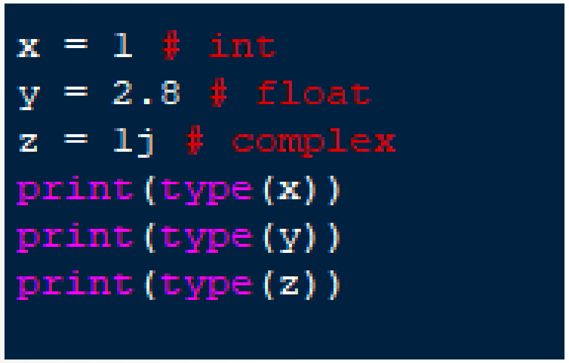
- Set Types: set
- Mapping Type: dict
- Sequence Types:
  - Tuple
  - list



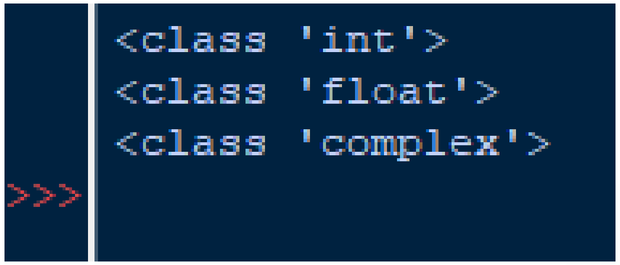
Numeric data types

There are three numeric types in Python:

- int
- float
- Complex



```
x = 1 # int
y = 2.8 # float
z = 1j # complex
print(type(x))
print(type(y))
print(type(z))
```

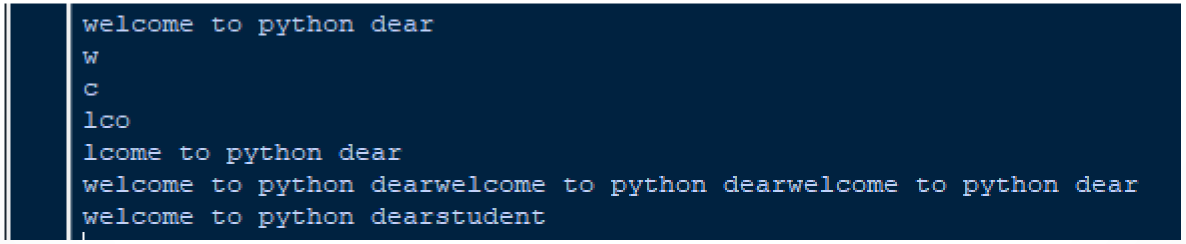


```
<class 'int'>
<class 'float'>
<class 'complex'>
>>>
```

String data type

- Strings in Python are identified as a **contiguous set of characters** represented in the **quotation marks.**
- Python allows for either pairs of
    - **Single quotes**
    - **double quotes**
- Subsets of strings can be taken using the **slice operator** ([]and[:])
- indexes starting at 0 in the beginning of the string
- working their way from-1 at the end.
- The plus (+) sign is string concatenation operator
- The asterisk (*) is the repetition operator.

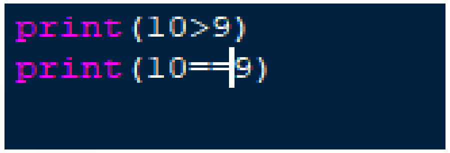
- str='welcome to python dear'
- print (str) #Prints complete string
- print (str[0]) #Prints first character of the string
- print (str[3]) #Prints fourth character of the string
- print (str[2:5]) #Prints characters starting from 3rd to 5th
- print (str[2:]) #Prints string starting from 3rd character to end
- print (str*3) #Prints string three times
- print (str+"student") #Prints concatenated string



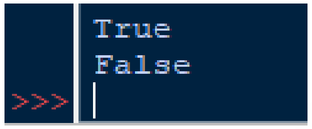
```
welcome to python dear
w
c
lco
lcome to python dear
welcome to python dearwelcome to python dearwelcome to python dear
welcome to python dearstudent
```

Boolean data type

- In programming you often need to know if an expression is
  - ➢ True
  - ➢ False.
- You can evaluate any expression in Python, and get one of two answers, True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer.



```
print(10>9)
print(10==9)
```



```
True
False
>>>
```

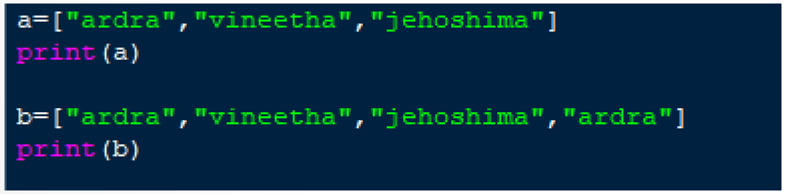
Sequence Types

	ordered	mutable	duplicate
list	ordered	mutable	Allow duplicate
tuple	ordered	Un mutable	Allow duplicate
dictionary	ordered	mutable	dictionaries do not allow duplicate keys, but they do allow duplicate values
set	Un ordered	mutable	Not allow duplicate values

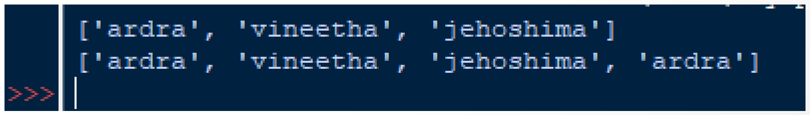
Sequence Types
List

- List is a collection which is ordered and changeable.
- Allows duplicate members.
- In Python lists are written with square brackets[].
- Allow Duplicates:

  Lists can have items with the same value:

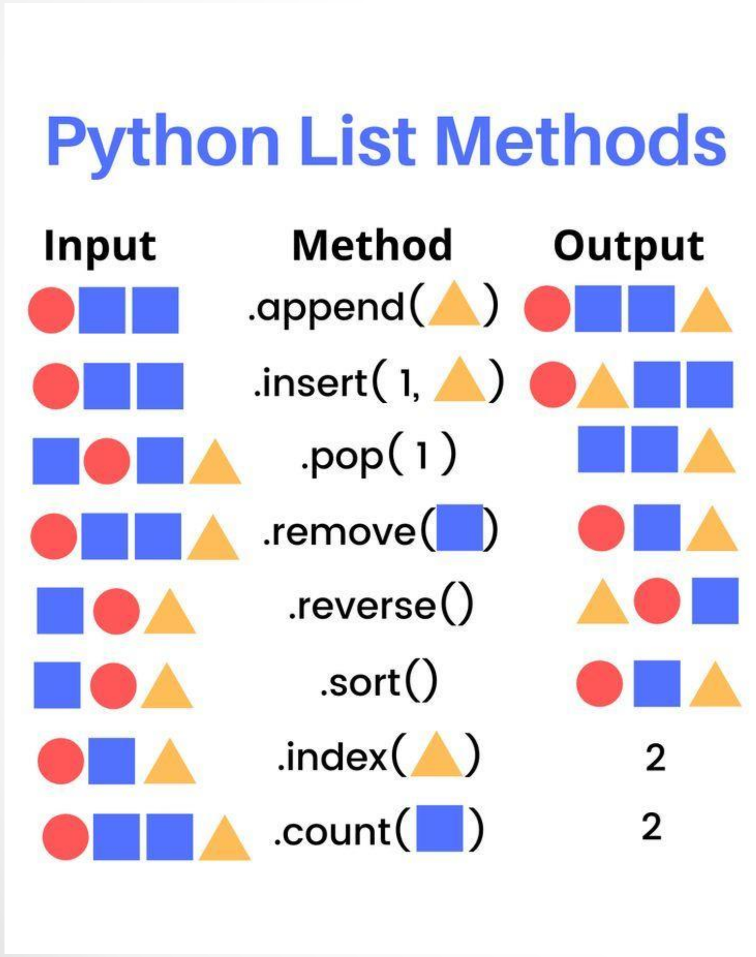


```
a=["ardra","vineetha","jehoshima"]
print(a)

b=["ardra","vineetha","jehoshima","ardra"]
print(b)
```



```
['ardra', 'vineetha', 'jehoshima']
['ardra', 'vineetha', 'jehoshima', 'ardra']
>>> |
```

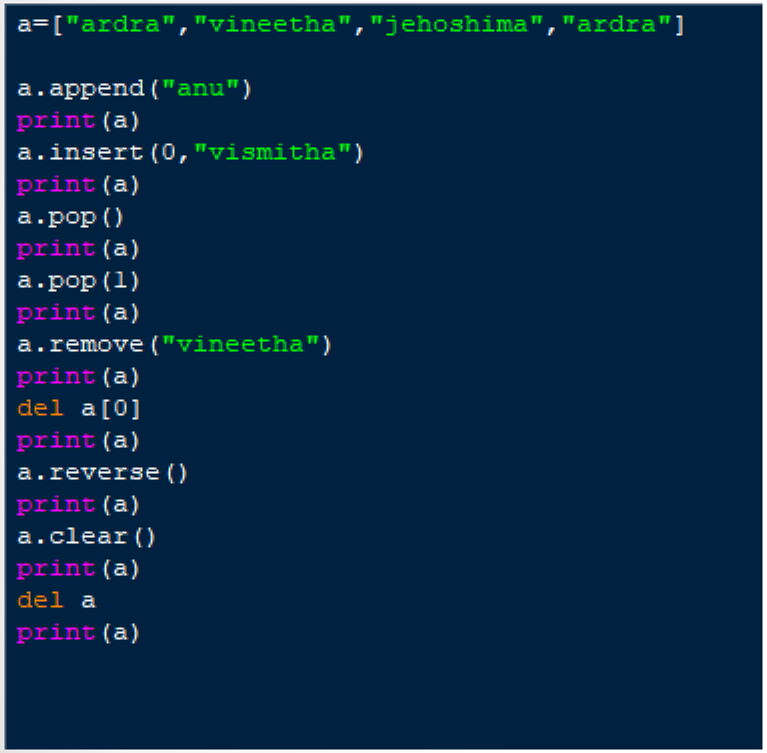
List methods



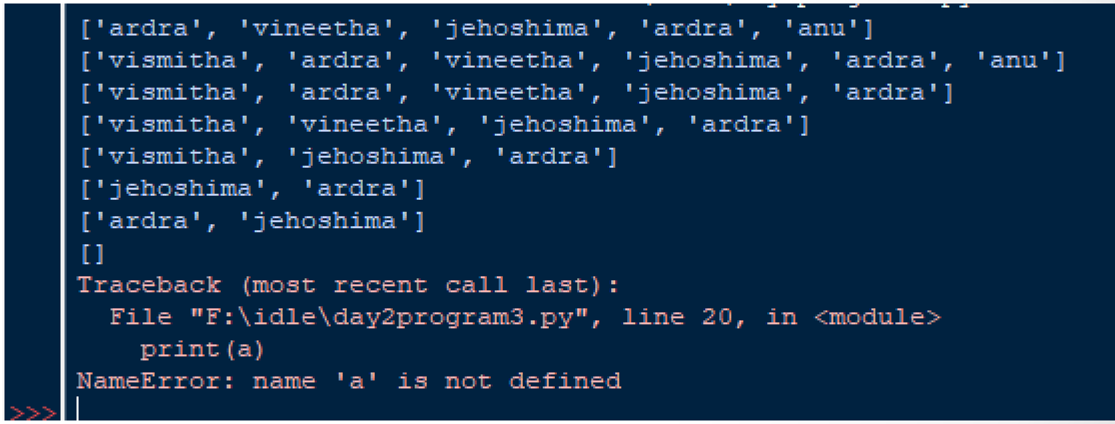
Python List Methods

Input	Method	Output
● ■ ■	.append(▲)	● ■ ■ ▲
● ■ ■	.insert(1, ▲)	● ▲ ■ ■
■ ● ■ ▲	.pop(1)	■ ■ ▲
● ■ ■ ▲	.remove(■)	● ■ ▲
■ ● ▲	.reverse()	▲ ● ■
■ ● ▲	.sort()	● ■ ▲
● ■ ▲	.index(▲)	2
● ■ ■ ▲	.count(■)	2

- **append()**- Add a item at end of the list .
- **insert -** Add a item in a specific position of list
- **pop()**
  - remove the specific Index
  - Removes last item if index not specified
- **remove()** - removes the specified item in list
- **clear()-** empties the list
- **del**
  - remove the specific Index
  - delete the list if index not specified



```python
a=["ardra","vineetha","jehoshima","ardra"]

a.append("anu")
print(a)
a.insert(0,"vismitha")
print(a)
a.pop()
print(a)
a.pop(1)
print(a)
a.remove("vineetha")
print(a)
del a[0]
print(a)
a.reverse()
print(a)
a.clear()
print(a)
del a
print(a)
```



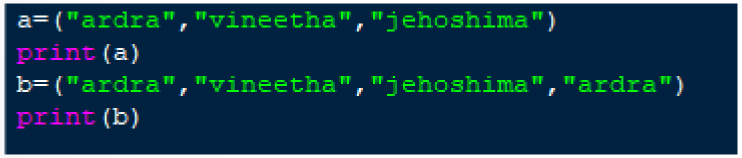
```
['ardra', 'vineetha', 'jehoshima', 'ardra', 'anu']
['vismitha', 'ardra', 'vineetha', 'jehoshima', 'ardra', 'anu']
['vismitha', 'ardra', 'vineetha', 'jehoshima', 'ardra']
['vismitha', 'vineetha', 'jehoshima', 'ardra']
['vismitha', 'jehoshima', 'ardra']
['jehoshima', 'ardra']
['ardra', 'jehoshima']
[]
Traceback (most recent call last):
  File "F:\idle\day2program3.py", line 20, in <module>
    print(a)
NameError: name 'a' is not defined
>>>
```

Tuples

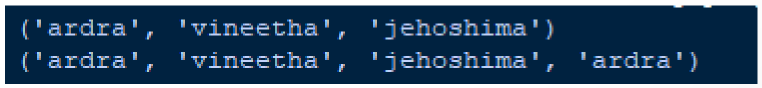
- A tuple is a collection which is ordered and unchangeable.
- Tuples allow duplicate values.
- Tuples are written with round brackets().

Unchangeable

- Sets are unchangeable, meaning that we cannot change the items after the set has been created.

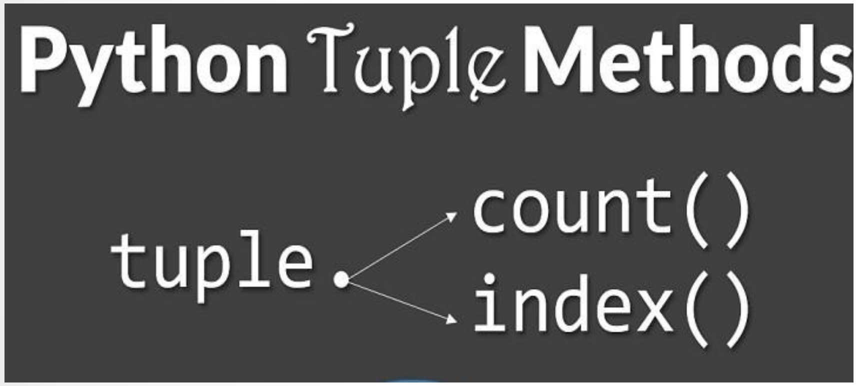


```
a=("ardra","vineetha","jehoshima")
print(a)
b=("ardra","vineetha","jehoshima","ardra")
print(b)
```



```
('ardra', 'vineetha', 'jehoshima')
('ardra', 'vineetha', 'jehoshima', 'ardra')
```

Tuple methods

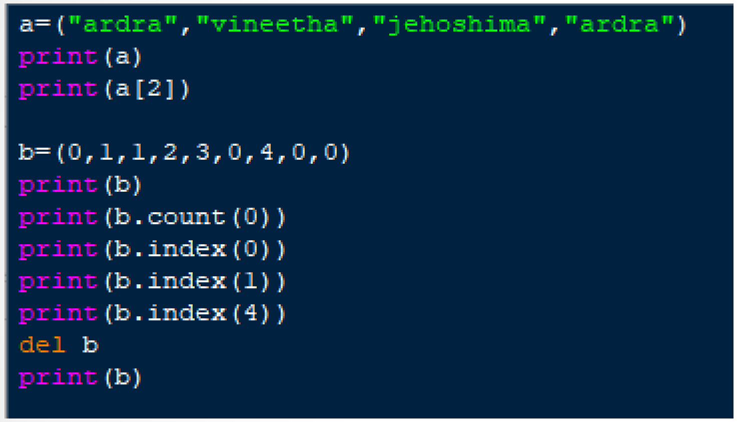


- **count()**

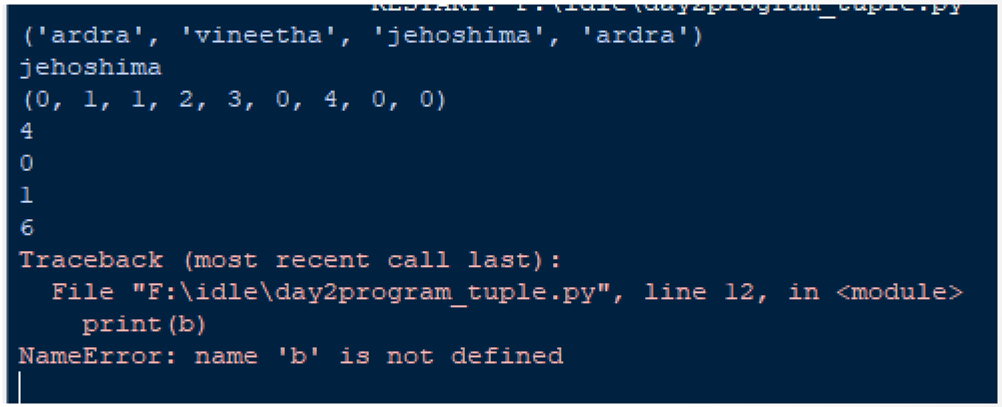
  The count() method returns the number of times a specified value appears in the tuple

- **index ()**

  The index() method finds the first occurrence of the specified value



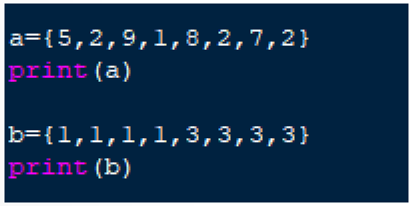
```
a=("ardra","vineetha","jehoshima","ardra")
print(a)
print(a[2])

b=(0,1,1,2,3,0,4,0,0)
print(b)
print(b.count(0))
print(b.index(0))
print(b.index(1))
print(b.index(4))
del b
print(b)
```



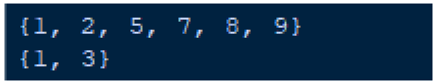
```
                        RESTART: F:\idle\day2program_tuple.py
('ardra', 'vineetha', 'jehoshima', 'ardra')
jehoshima
(0, 1, 1, 2, 3, 0, 4, 0, 0)
4
0
1
6
Traceback (most recent call last):
  File "F:\idle\day2program_tuple.py", line 12, in <module>
    print(b)
NameError: name 'b' is not defined
```

Set

- Set is a collection which is unordered and changeable .
- **Not allow duplicate value.**
- In Python set are written with curly brackets {} .



```
a={5,2,9,1,8,2,7,2}
print(a)

b={1,1,1,1,3,3,3,3}
print(b)
```

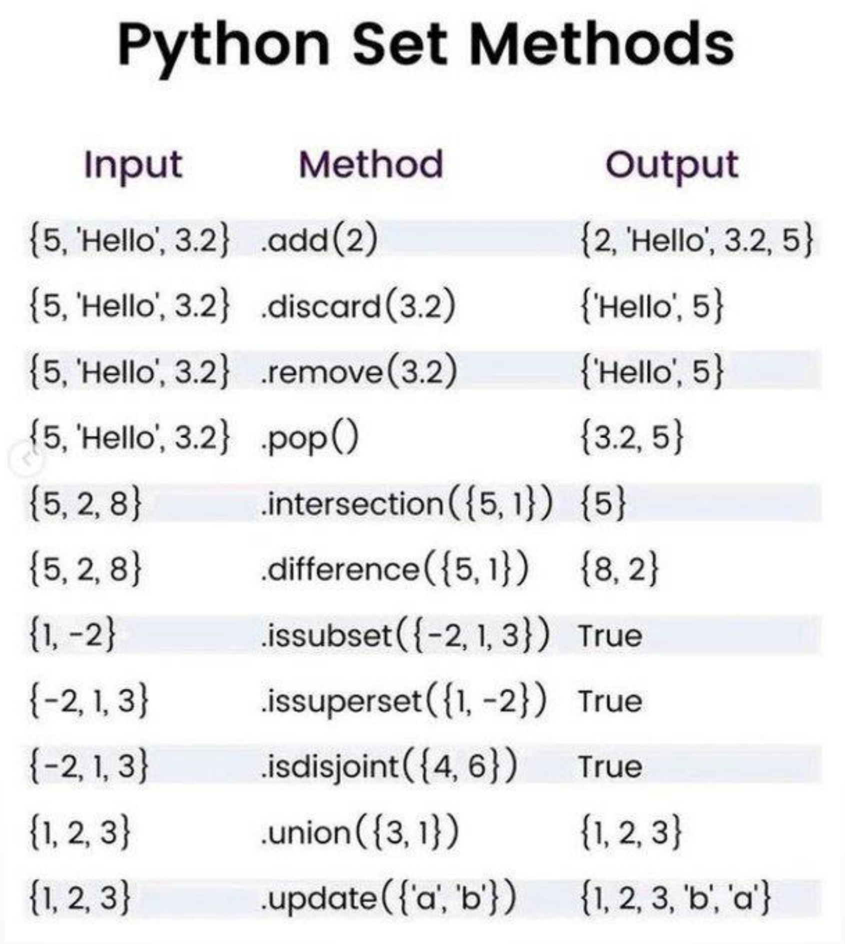


```
{1, 2, 5, 7, 8, 9}
{1, 3}
```

Unordered

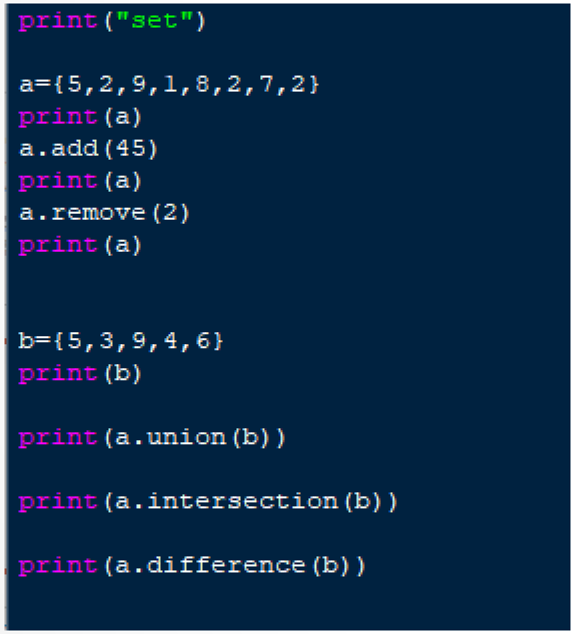
- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Set methods

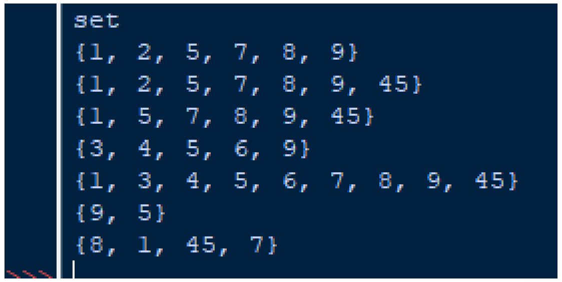


Python Set Methods

Input	Method	Output
{5, 'Hello', 3.2}	.add(2)	{2, 'Hello', 3.2, 5}
{5, 'Hello', 3.2}	.discard(3.2)	{'Hello', 5}
{5, 'Hello', 3.2}	.remove(3.2)	{'Hello', 5}
{5, 'Hello', 3.2}	.pop()	{3.2, 5}
{5, 2, 8}	.intersection({5, 1})	{5}
{5, 2, 8}	.difference({5, 1})	{8, 2}
{1, -2}	.issubset({-2, 1, 3})	True
{-2, 1, 3}	.issuperset({1, -2})	True
{-2, 1, 3}	.isdisjoint({4, 6})	True
{1, 2, 3}	.union({3, 1})	{1, 2, 3}
{1, 2, 3}	.update({'a', 'b'})	{1, 2, 3, 'b', 'a'}



```
print("set")

a={5,2,9,1,8,2,7,2}
print(a)
a.add(45)
print(a)
a.remove(2)
print(a)


b={5,3,9,4,6}
print(b)

print(a.union(b))

print(a.intersection(b))

print(a.difference(b))
```



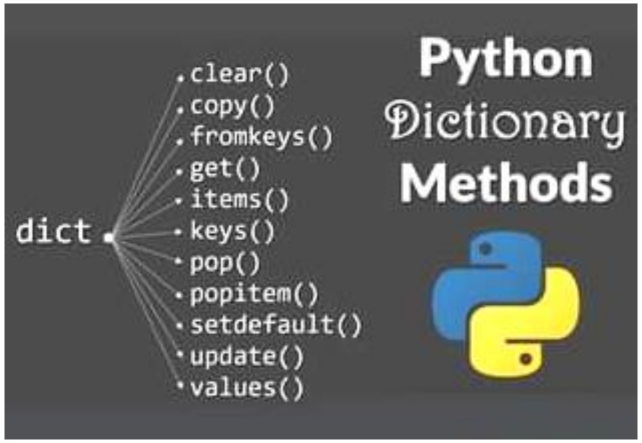
```
set
{1, 2, 5, 7, 8, 9}
{1, 2, 5, 7, 8, 9, 45}
{1, 5, 7, 8, 9, 45}
{3, 4, 5, 6, 9}
{1, 3, 4, 5, 6, 7, 8, 9, 45}
{9, 5}
{8, 1, 45, 7}
```

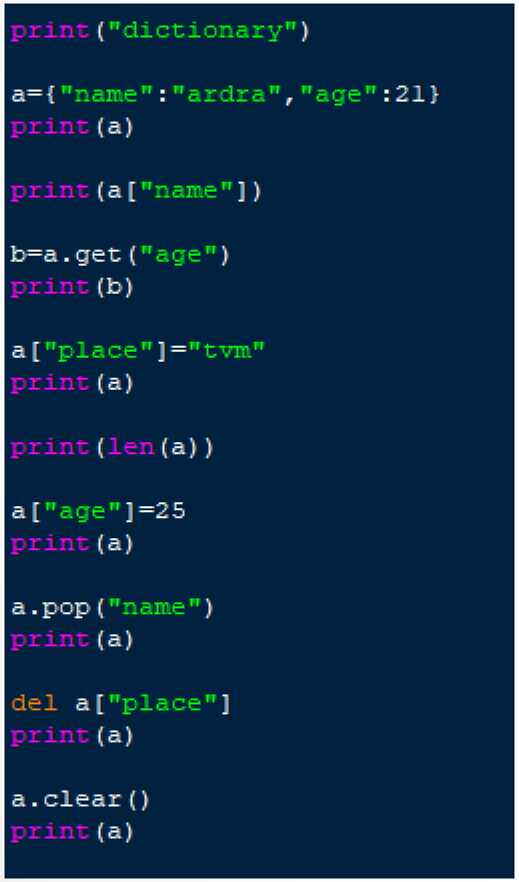
DICTIONARY

- Dictionary is a collection which is ordered and changeable.
- No duplicate members.
- In Python dictionary are written with curly brackets and they keys and values

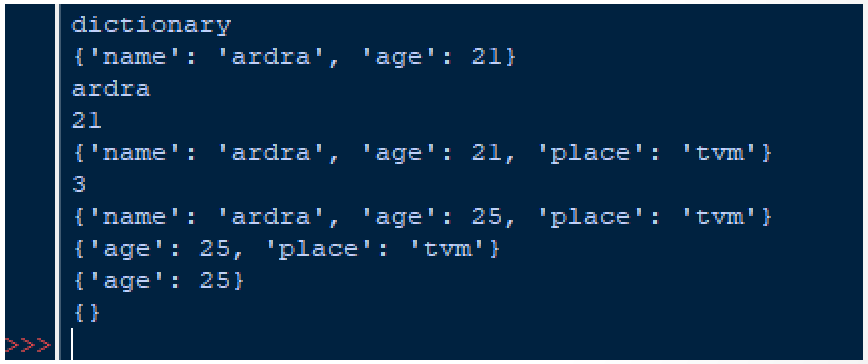
{‘key’:value}

Dictionary methods



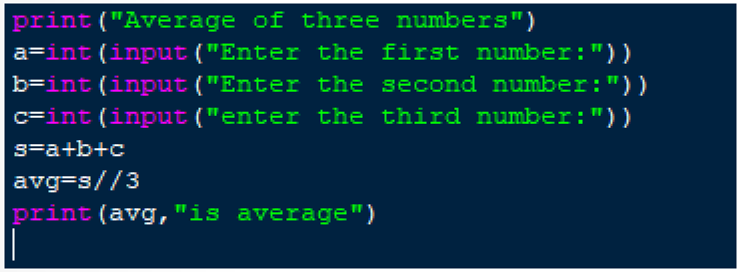


```
print("dictionary")

a={"name":"ardra","age":21}
print(a)

print(a["name"])

b=a.get("age")
print(b)

a["place"]="tvm"
print(a)

print(len(a))

a["age"]=25
print(a)

a.pop("name")
print(a)

del a["place"]
print(a)

a.clear()
print(a)
```

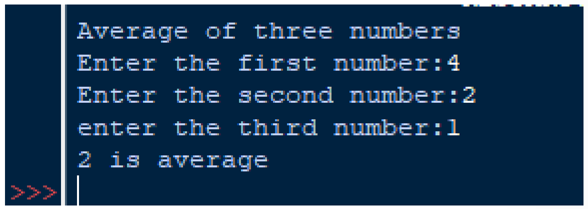


```
dictionary
{'name': 'ardra', 'age': 21}
ardra
21
{'name': 'ardra', 'age': 21, 'place': 'tvm'}
3
{'name': 'ardra', 'age': 25, 'place': 'tvm'}
{'age': 25, 'place': 'tvm'}
{'age': 25}
{}
>>>
```

- Find average of three numbers

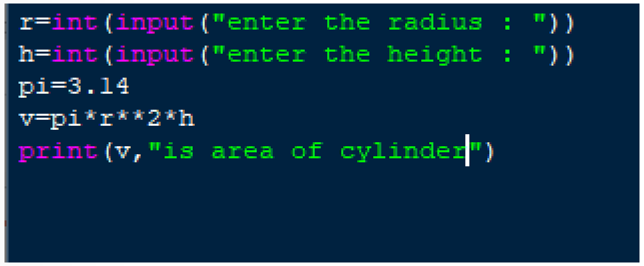


```python
print("Average of three numbers")
a=int(input("Enter the first number:"))
b=int(input("Enter the second number:"))
c=int(input("enter the third number:"))
s=a+b+c
avg=s//3
print(avg,"is average")
```

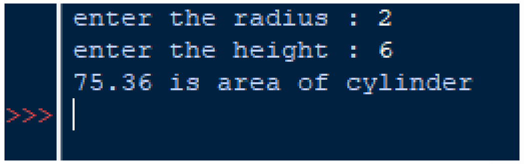


```
Average of three numbers
Enter the first number:4
Enter the second number:2
enter the third number:1
2 is average
>>>
```

- Find the volume of a cylinder



```
r=int(input("enter the radius : "))
h=int(input("enter the height : "))
pi=3.14
v=pi*r**2*h
print(v,"is area of cylinder")
```



```
enter the radius : 2
enter the height : 6
75.36 is area of cylinder
>>>
```

PYTHON CONDITIONS & IF-STATEMENTS

IF-STATEMENTS

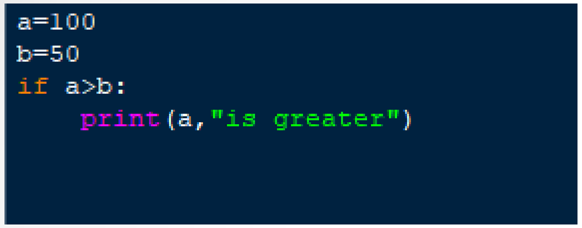
Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

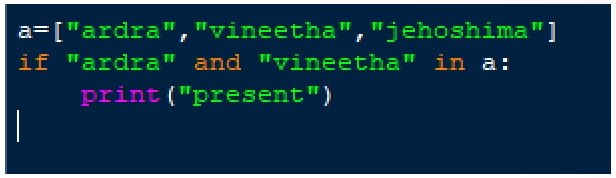
These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

- if



```
a=100
b=50
if a>b:
    print(a,"is greater")
```



```
a=["ardra","vineetha","jehoshima"]
if "ardra" and "vineetha" in a:
    print("present")
```

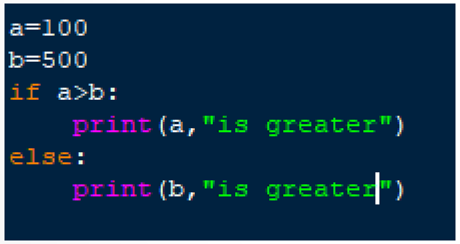
output

100 is greater

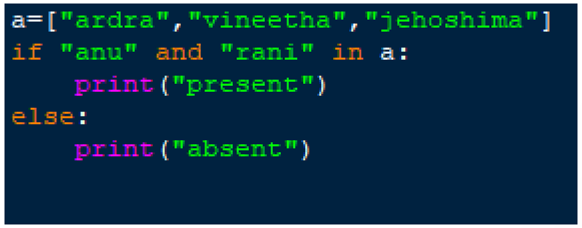
output

present

- if else



```
a=100
b=500
if a>b:
    print(a,"is greater")
else:
    print(b,"is greater")
```



```
a=["ardra","vineetha","jehoshima"]
if "anu" and "rani" in a:
    print("present")
else:
    print("absent")
```

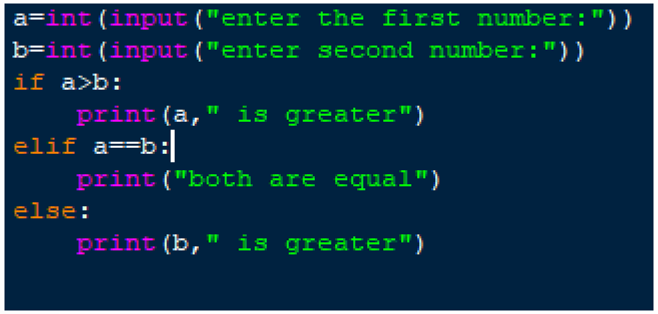
output

500 is greater

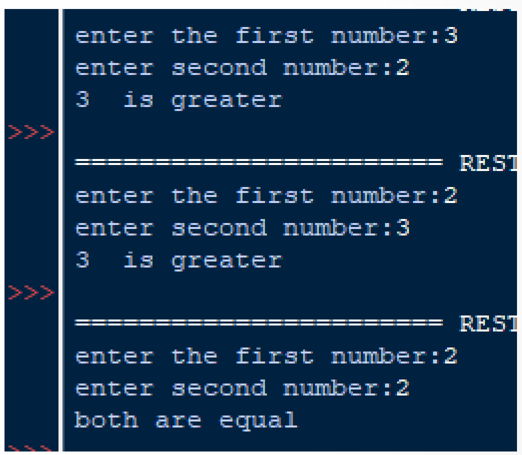
output

abesent

- if , elif and else



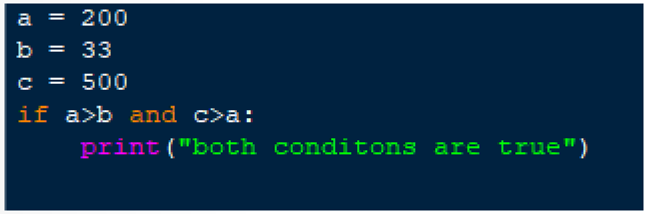
```
a=int(input("enter the first number:"))
b=int(input("enter second number:"))
if a>b:
    print(a," is greater")
elif a==b:
    print("both are equal")
else:
    print(b," is greater")
```



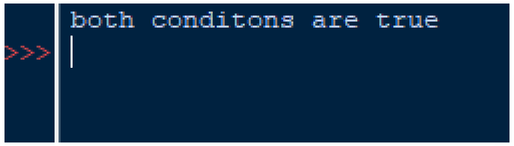
```
    enter the first number:3
    enter second number:2
    3  is greater
>>>
    ======================== REST
    enter the first number:2
    enter second number:3
    3  is greater
>>>
    ======================== REST
    enter the first number:2
    enter second number:2
    both are equal
```

and

- The **and** keyword is a logical operator
- It is used to combine conditional statements



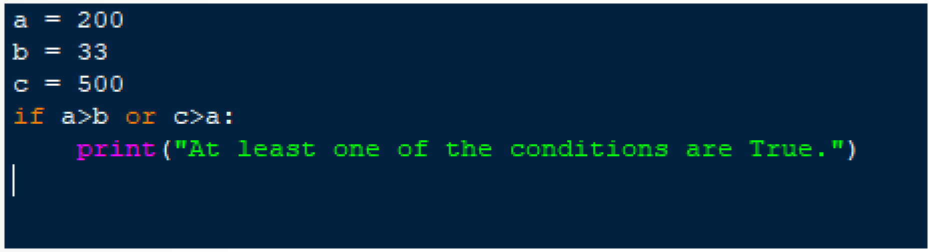
```
a = 200
b = 33
c = 500
if a>b and c>a:
    print("both conditons are true")
```



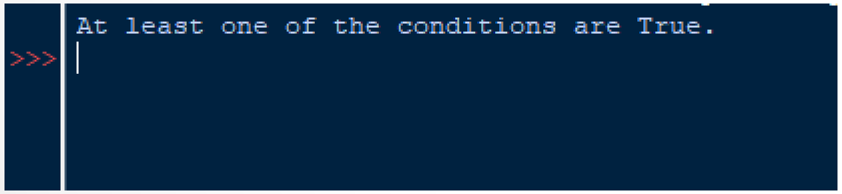
```
both conditons are true
>>> |
```

or

- **or** keyword is a logical operator
- It is used to combine conditional statements

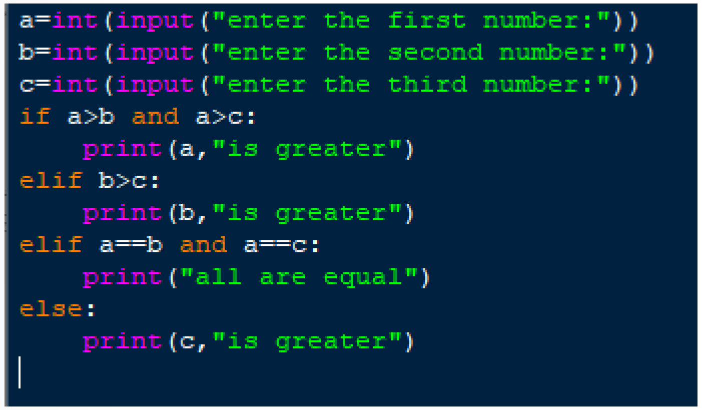


```
a = 200
b = 33
c = 500
if a>b or c>a:
    print("At least one of the conditions are True.")
```

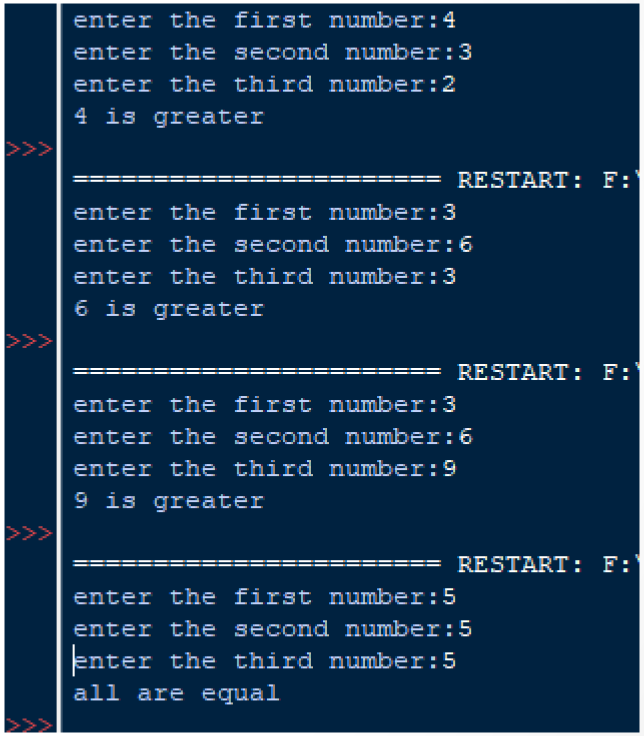


```
    At least one of the conditions are True.
>>> 
```

1.Largest of three numbers

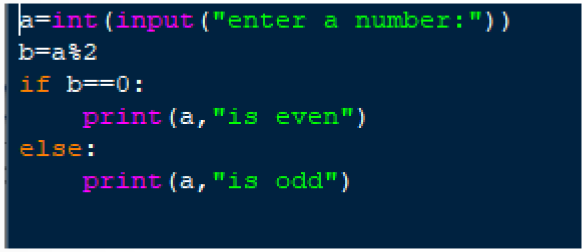


```python
a=int(input("enter the first number:"))
b=int(input("enter the second number:"))
c=int(input("enter the third number:"))
if a>b and a>c:
    print(a,"is greater")
elif b>c:
    print(b,"is greater")
elif a==b and a==c:
    print("all are equal")
else:
    print(c,"is greater")
```

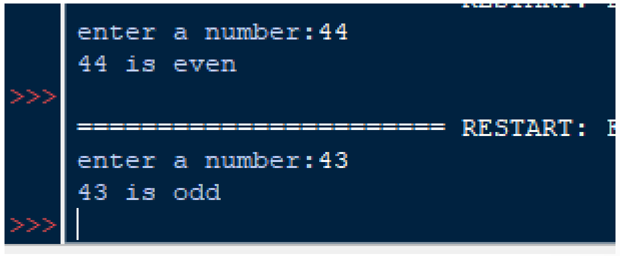


```
    enter the first number:4
    enter the second number:3
    enter the third number:2
    4 is greater
>>> 
    ======================= RESTART: F:
    enter the first number:3
    enter the second number:6
    enter the third number:3
    6 is greater
>>> 
    ======================= RESTART: F:
    enter the first number:3
    enter the second number:6
    enter the third number:9
    9 is greater
>>> 
    ======================= RESTART: F:
    enter the first number:5
    enter the second number:5
    enter the third number:5
    all are equal
>>> 
```

2. Find a number is Odd or even

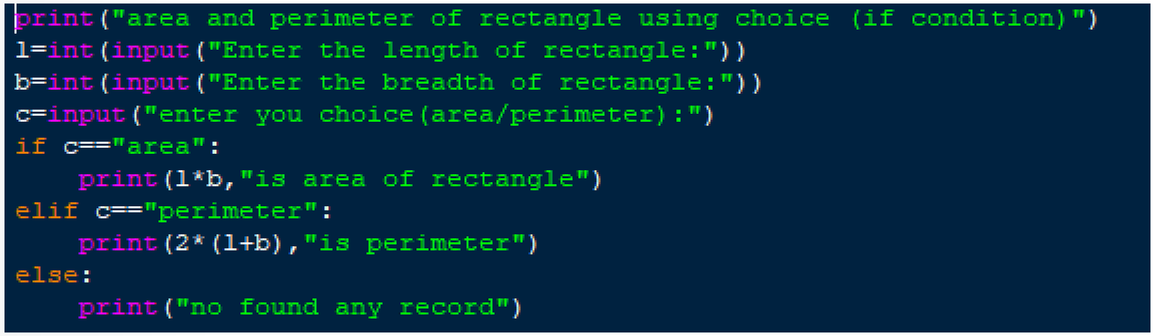


```
a=int(input("enter a number:"))
b=a%2
if b==0:
    print(a,"is even")
else:
    print(a,"is odd")
```

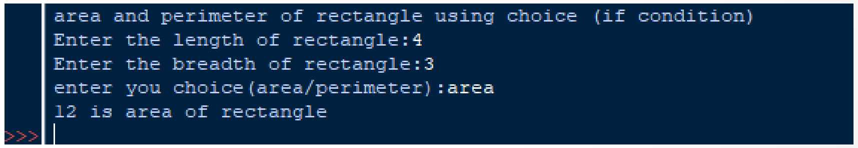


```
enter a number:44
44 is even
>>>
======================== RESTART:
enter a number:43
43 is odd
>>>
```

3.Find area and perimeter of rectangle using if , elif and else

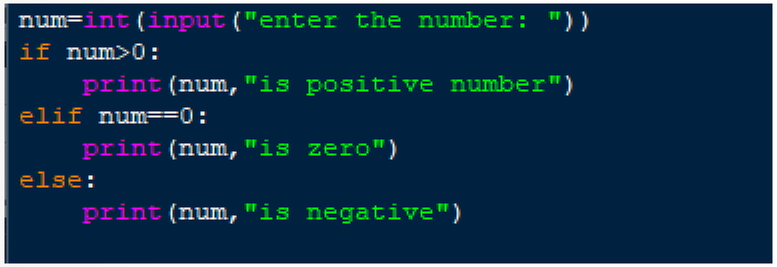


```
print("area and perimeter of rectangle using choice (if condition)")
l=int(input("Enter the length of rectangle:"))
b=int(input("Enter the breadth of rectangle:"))
c=input("enter you choice(area/perimeter):")
if c=="area":
    print(l*b,"is area of rectangle")
elif c=="perimeter":
    print(2*(l+b),"is perimeter")
else:
    print("no found any record")
```

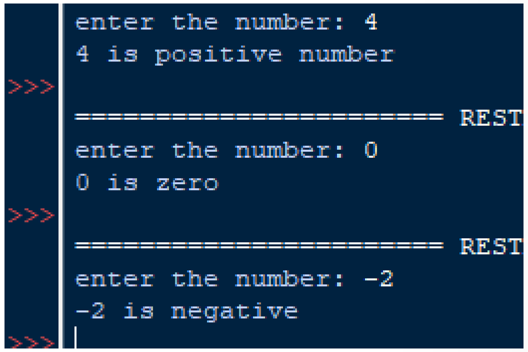


```
area and perimeter of rectangle using choice (if condition)
Enter the length of rectangle:4
Enter the breadth of rectangle:3
enter you choice(area/perimeter):area
12 is area of rectangle
>>> 
```

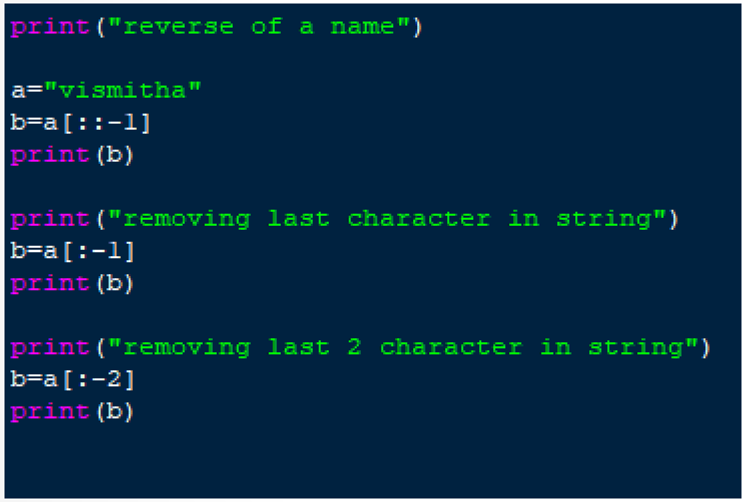
4.find the number is positive negative or zero



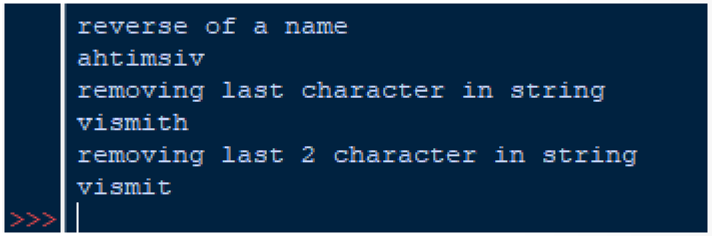
```
num=int(input("enter the number: "))
if num>0:
    print(num,"is positive number")
elif num==0:
    print(num,"is zero")
else:
    print(num,"is negative")
```



```
    enter the number: 4
    4 is positive number
>>>
    ======================= REST
    enter the number: 0
    0 is zero
>>>
    ======================= REST
    enter the number: -2
    -2 is negative
>>> 
```

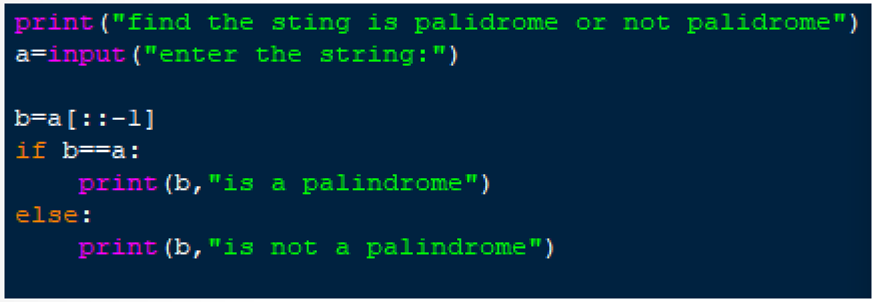


```
print("reverse of a name")

a="vismitha"
b=a[::-1]
print(b)

print("removing last character in string")
b=a[:-1]
print(b)

print("removing last 2 character in string")
b=a[:-2]
print(b)
```

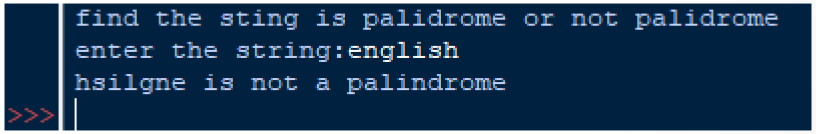


```
reverse of a name
ahtimsiv
removing last character in string
vismith
removing last 2 character in string
vismit
>>>
```

5.Find a String is Palindrome or Not

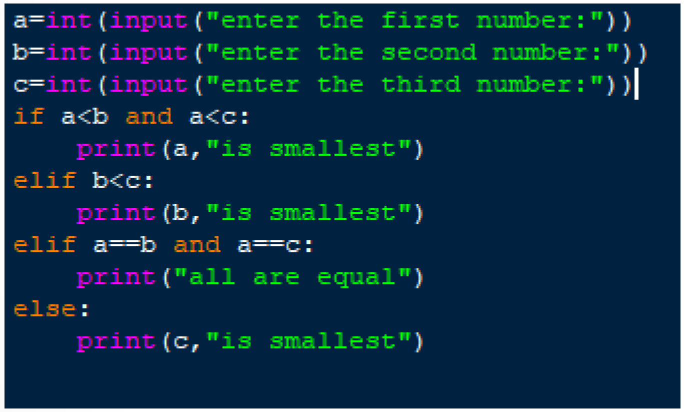


```
print("find the sting is palidrome or not palidrome")
a=input("enter the string:")

b=a[::-1]
if b==a:
    print(b,"is a palindrome")
else:
    print(b,"is not a palindrome")
```

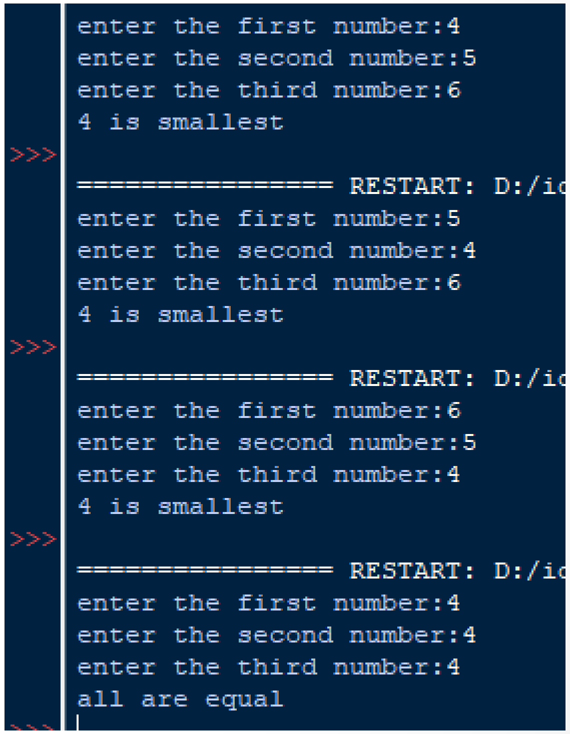


```
find the sting is palidrome or not palidrome
enter the string:english
hsilgne is not a palindrome
>>> 
```

- Find Smallest of three numbers.



```
a=int(input("enter the first number:"))
b=int(input("enter the second number:"))
c=int(input("enter the third number:"))
if a<b and a<c:
    print(a,"is smallest")
elif b<c:
    print(b,"is smallest")
elif a==b and a==c:
    print("all are equal")
else:
    print(c,"is smallest")
```



```
    enter the first number:4
    enter the second number:5
    enter the third number:6
    4 is smallest
>>>
    ================ RESTART: D:/i
    enter the first number:5
    enter the second number:4
    enter the third number:6
    4 is smallest
>>>
    ================ RESTART: D:/i
    enter the first number:6
    enter the second number:5
    enter the third number:4
    4 is smallest
>>>
    ================ RESTART: D:/i
    enter the first number:4
    enter the second number:4
    enter the third number:4
    all are equal
```

PYTHON LOOPS

Python has two primitive loop commands:

- ➢**while loops**
- ➢**for loops**

For Loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
#single range
for x in range(6):
    print(x)

#variable
a=10
for i in range(a+1):
    print(i)

#range starting value to
ending value
for x in range(2, 6):
    print(x)
```

```
#incrementation parameter
for x in range(2, 30, 3):
    print(x)

#decrementation parameter
for x in range(30,2, -3):
    print(x)

#for in a list
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
#loop through a string
for x in "banana":
    print(x)
```

```
#loop through a string with
variable
a="welcome to python shell"
for x in a:
    print (x)
```

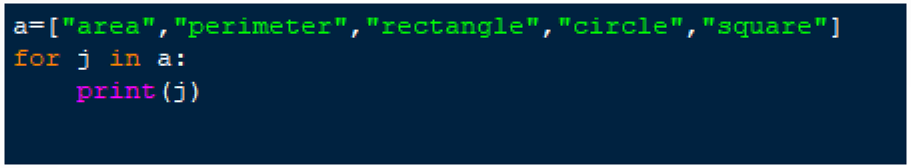
```
#for else
for x in range(6):
    print(x)
else:
    print("finally finished!")
```

```
#exit the loop when x is "banana":
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
    print(x)
```

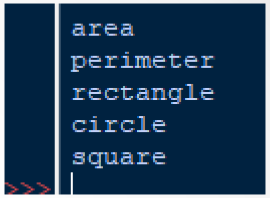
```
#exit the loop when x is "banana"
#but this time the break comes
before the print:
#break statement in a list of items
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        break
    print(x)
```

```
#pass statement
for x in [0, 1, 2]:
    pass
```

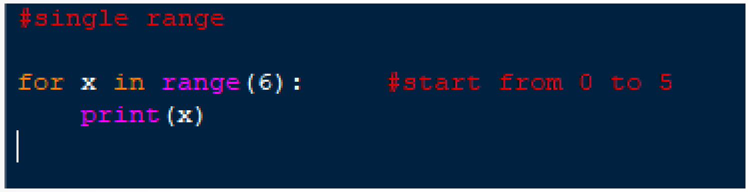
```
#nested loops
properties= ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in properties:
    for y in fruits:
        print(x, y)
```



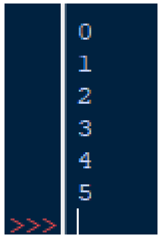
```
a=["area","perimeter","rectangle","circle","square"]
for j in a:
    print(j)
```



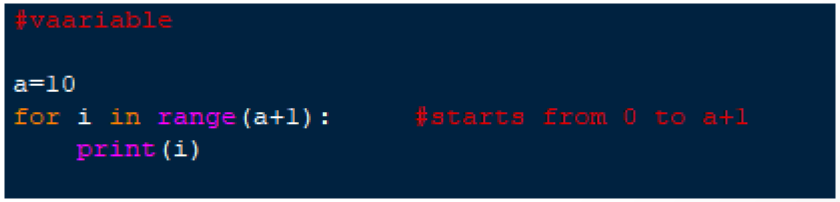
```
area
perimeter
rectangle
circle
square
>>>
```



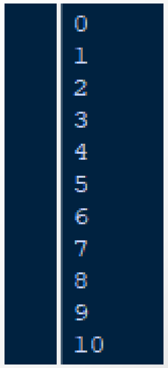
```
#single range

for x in range(6):      #start from 0 to 5
    print(x)
```



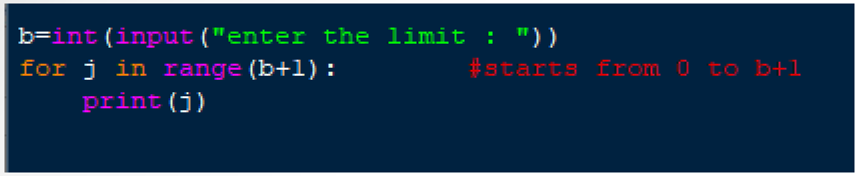
```
0
1
2
3
4
5
>>>
```



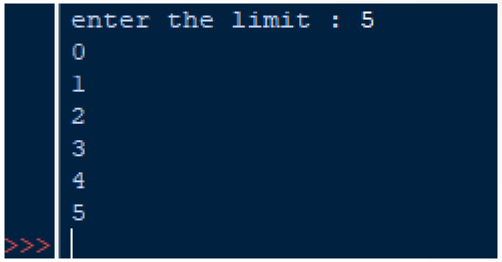
```
#vaariable

a=10
for i in range(a+1):      #starts from 0 to a+1
    print(i)
```



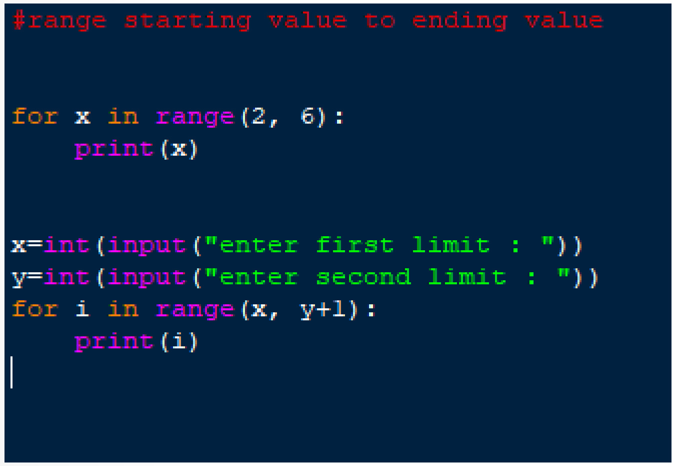
```
0
1
2
3
4
5
6
7
8
9
10
```



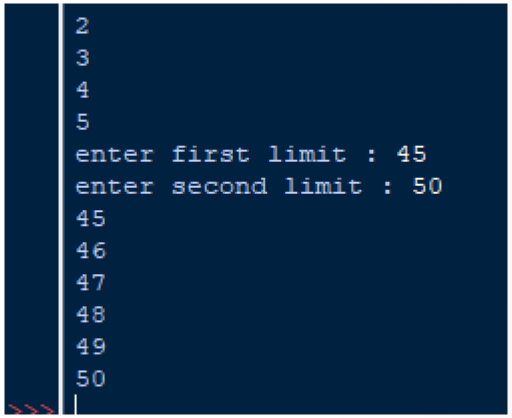
```
b=int(input("enter the limit : "))
for j in range(b+1):          #starts from 0 to b+1
    print(j)
```



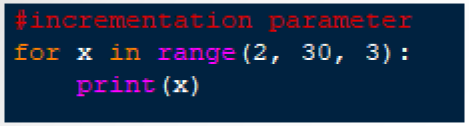
```
enter the limit : 5
0
1
2
3
4
5
>>>
```



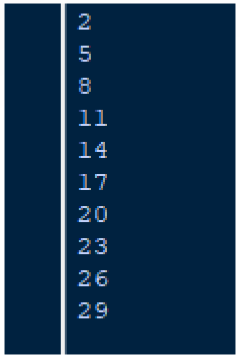
```
#range starting value to ending value


for x in range(2, 6):
    print(x)


x=int(input("enter first limit : "))
y=int(input("enter second limit : "))
for i in range(x, y+1):
    print(i)
```



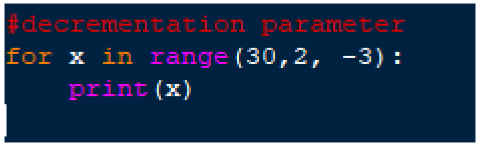
```
2
3
4
5
enter first limit : 45
enter second limit : 50
45
46
47
48
49
50
>>>
```



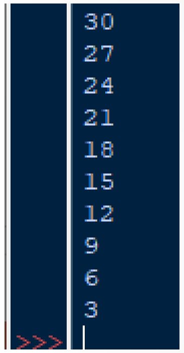
```
#incrementation parameter
for x in range(2, 30, 3):
    print(x)
```



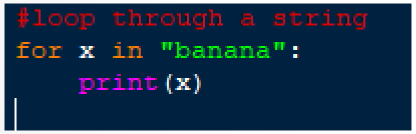
```
2
5
8
11
14
17
20
23
26
29
```



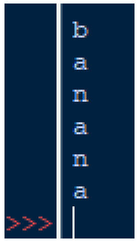
```
#decrementation parameter
for x in range(30,2, -3):
    print(x)
```



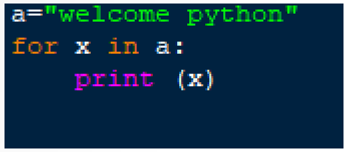
```
30
27
24
21
18
15
12
9
6
3
>>>
```



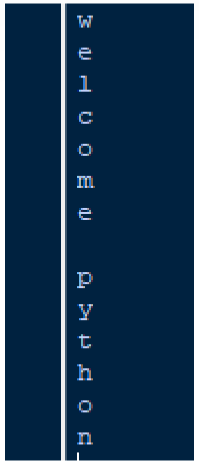
```
#loop through a string
for x in "banana":
    print(x)
```



```
b
a
n
a
n
a
>>>
```



```
a="welcome python"
for x in a:
    print (x)
```



```
w
e
l
c
o
m
e
 
p
y
t
h
o
n
```

What does end =′ do in Python?

- The end parameter in the print function is used to add any string.
- At the end of the output of the print statement in python.
- By default, the print function ends with a newline.
- Passing the whitespace to the end parameter (end=‘ ‘) indicates that the end character has to be identified by whitespace and not a newline.

Example:

```
print(“hello”, end=′ ‘)
print(“welcome to python”)
```

Output:

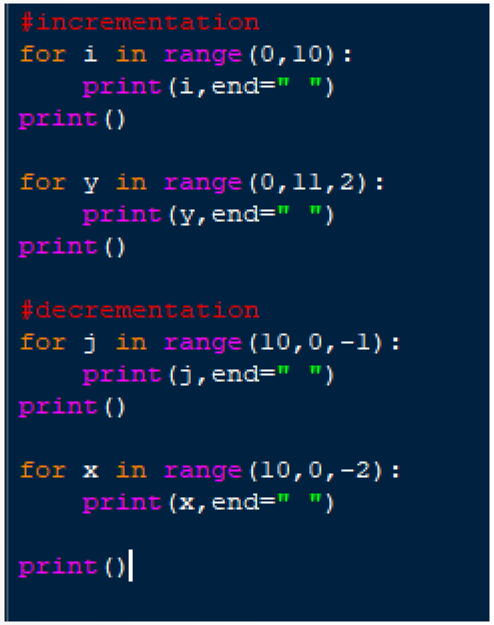
```
Hello welcome to python
```

Example:

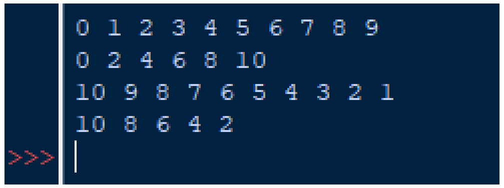
```
for j in range(0,5):
    print(j,end=" “)
```

Output:

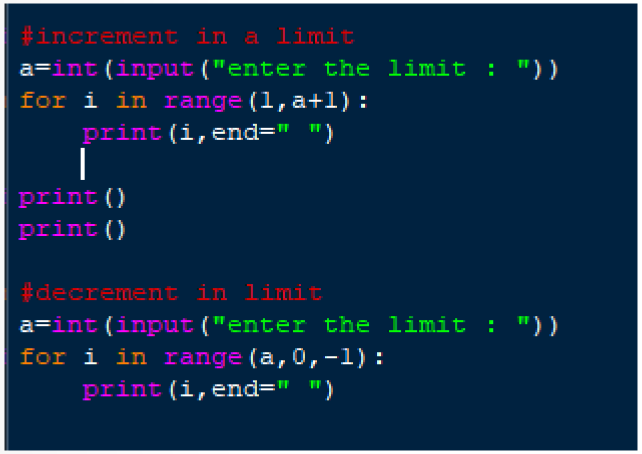
```
0 1 2 3 4
```



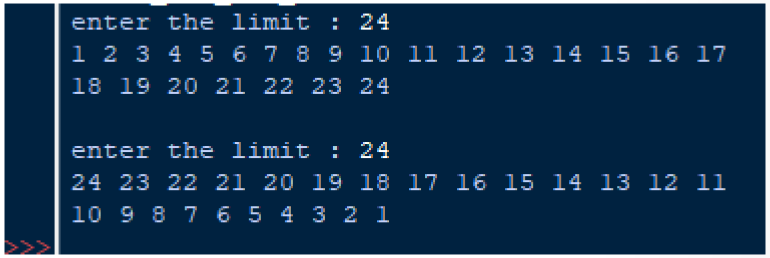
```
#incrementation
for i in range(0,10):
    print(i,end=" ")
print()

for y in range(0,11,2):
    print(y,end=" ")
print()

#decrementation
for j in range(10,0,-1):
    print(j,end=" ")
print()

for x in range(10,0,-2):
    print(x,end=" ")

print()
```



```
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8 10
10 9 8 7 6 5 4 3 2 1
10 8 6 4 2
>>>
```

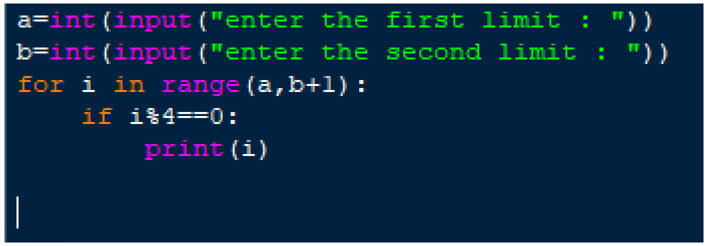


```
#increment in a limit
a=int(input("enter the limit : "))
for i in range(1,a+1):
    print(i,end=" ")

print()
print()

#decrement in limit
a=int(input("enter the limit : "))
for i in range(a,0,-1):
    print(i,end=" ")
```

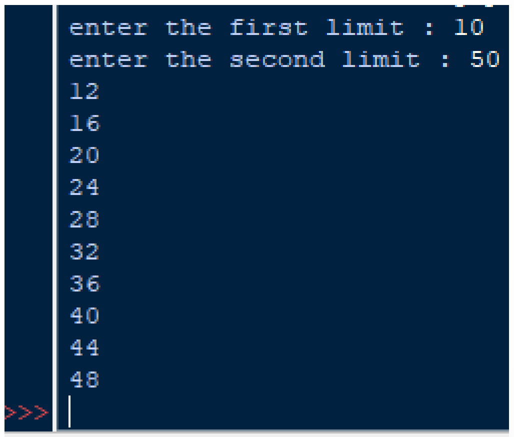


```
enter the limit : 24
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24

enter the limit : 24
24 23 22 21 20 19 18 17 16 15 14 13 12 11
10 9 8 7 6 5 4 3 2 1
>>>
```

- Print the multiple of 4 with in a range

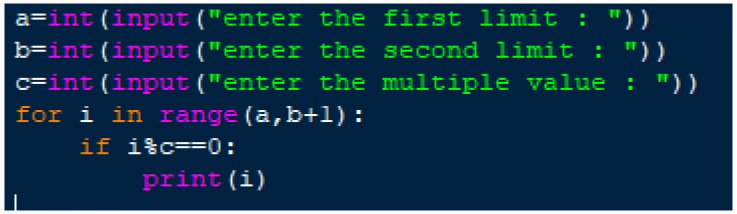


```
a=int(input("enter the first limit : "))
b=int(input("enter the second limit : "))
for i in range(a,b+1):
    if i%4==0:
        print(i)
```

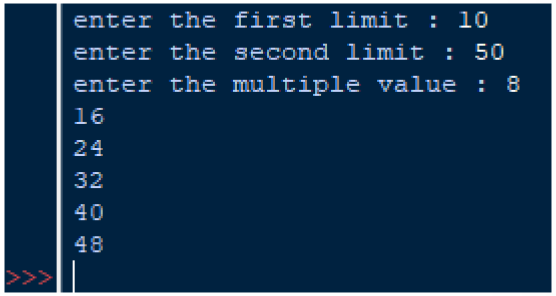


```
enter the first limit : 10
enter the second limit : 50
12
16
20
24
28
32
36
40
44
48
>>>
```

- Enter the multiple and print it in a limit

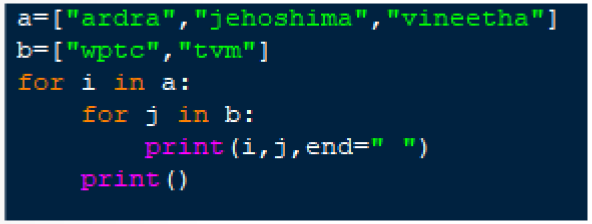


```
a=int(input("enter the first limit : "))
b=int(input("enter the second limit : "))
c=int(input("enter the multiple value : "))
for i in range(a,b+1):
    if i%c==0:
        print(i)
```

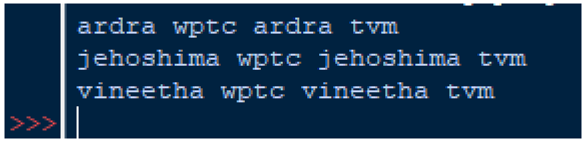


```
enter the first limit : 10
enter the second limit : 50
enter the multiple value : 8
16
24
32
40
48
>>>
```

Nested for

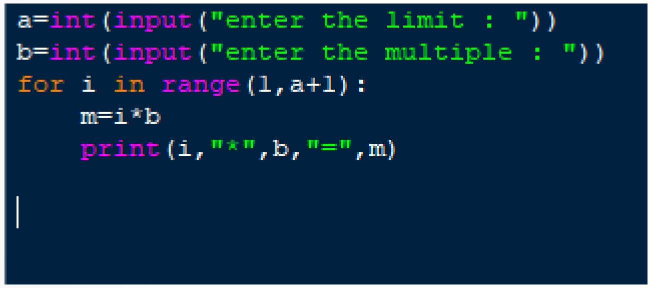


```
a=["ardra","jehoshima","vineetha"]
b=["wptc","tvm"]
for i in a:
    for j in b:
        print(i,j,end=" ")
    print()
```

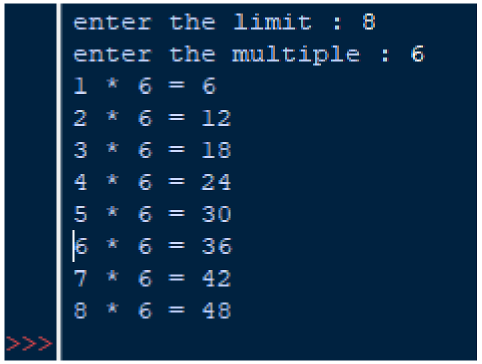


```
ardra wptc ardra tvm
jehoshima wptc jehoshima tvm
vineetha wptc vineetha tvm
>>> 
```

- Multiplication table

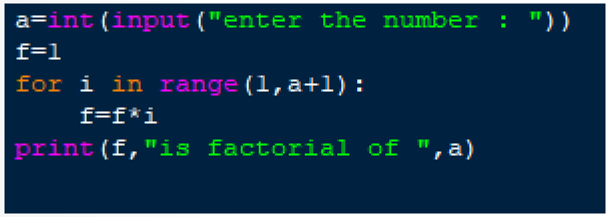


```
a=int(input("enter the limit : "))
b=int(input("enter the multiple : "))
for i in range(1,a+1):
    m=i*b
    print(i,"*",b,"=",m)
```

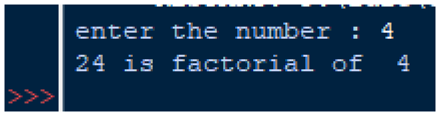


```
enter the limit : 8
enter the multiple : 6
1 * 6 = 6
2 * 6 = 12
3 * 6 = 18
4 * 6 = 24
5 * 6 = 30
6 * 6 = 36
7 * 6 = 42
8 * 6 = 48
>>>
```

Find Factorial of a number a

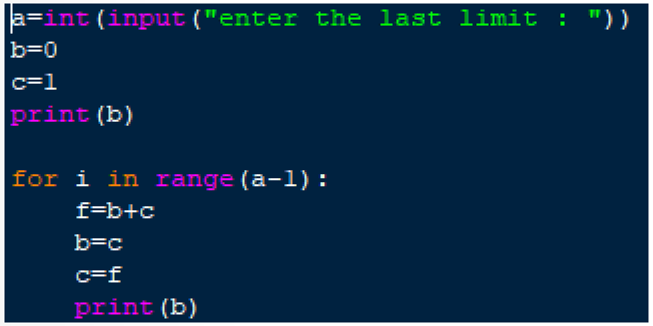


```
a=int(input("enter the number : "))
f=1
for i in range(1,a+1):
    f=f*i
print(f,"is factorial of ",a)
```

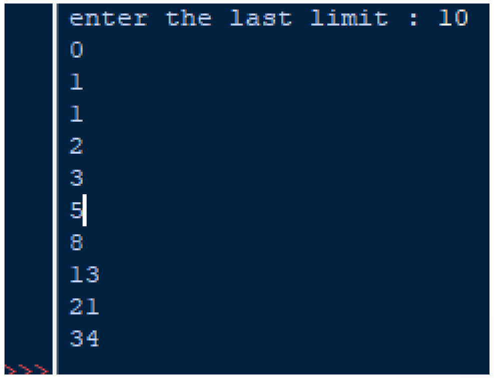


```
enter the number : 4
24 is factorial of  4
>>>
```

- Fibonacci Series

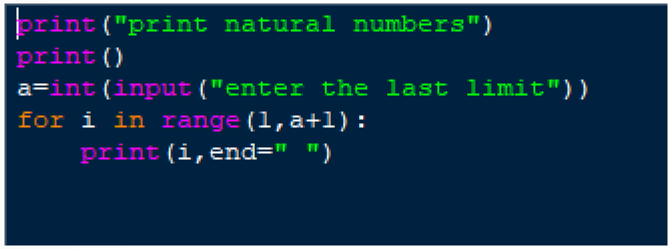


```
a=int(input("enter the last limit : "))
b=0
c=1
print(b)

for i in range(a-1):
    f=b+c
    b=c
    c=f
    print(b)
```

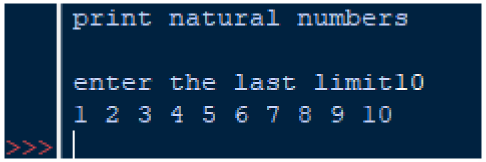


```
enter the last limit : 10
0
1
1
2
3
5
8
13
21
34
>>>
```

- Print natural numbers

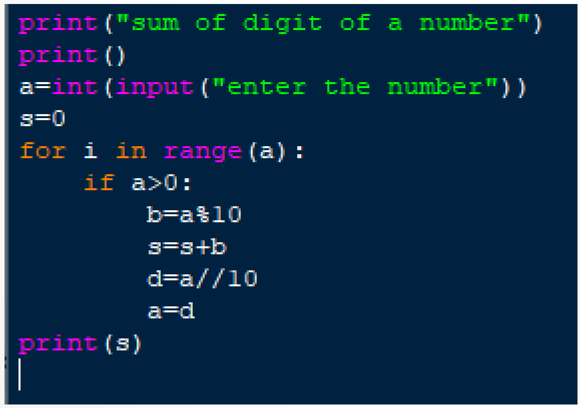


```
print("print natural numbers")
print()
a=int(input("enter the last limit"))
for i in range(1,a+1):
    print(i,end=" ")
```

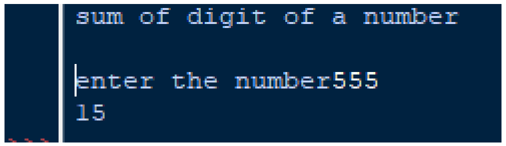


```
print natural numbers

enter the last limit10
1 2 3 4 5 6 7 8 9 10
>>>
```

- Sum of digits of a number

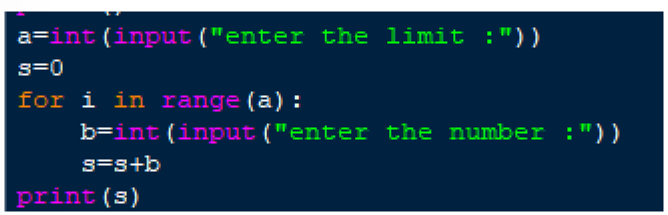


```
print("sum of digit of a number")
print()
a=int(input("enter the number"))
s=0
for i in range(a):
    if a>0:
        b=a%10
        s=s+b
        d=a//10
        a=d
print(s)
```

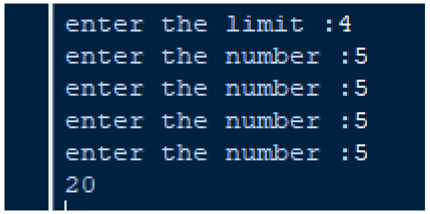


```
sum of digit of a number

enter the number555
15
```

- Sum of numbers

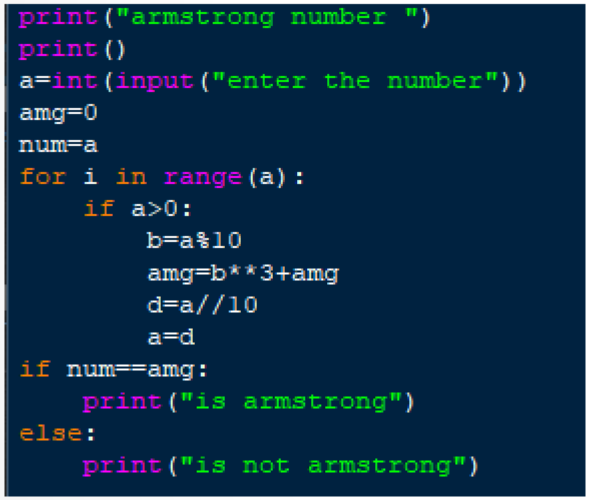


```
a=int(input("enter the limit :"))
s=0
for i in range(a):
    b=int(input("enter the number :"))
    s=s+b
print(s)
```

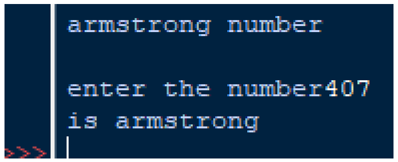


```
enter the limit :4
enter the number :5
enter the number :5
enter the number :5
enter the number :5
20
```

- Armstrong number

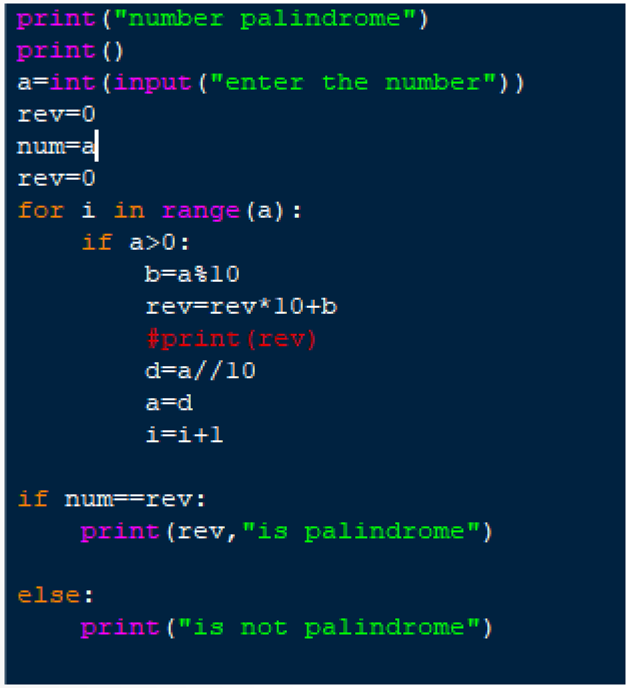


```
print("armstrong number ")
print()
a=int(input("enter the number"))
amg=0
num=a
for i in range(a):
    if a>0:
        b=a%10
        amg=b**3+amg
        d=a//10
        a=d
if num==amg:
    print("is armstrong")
else:
    print("is not armstrong")
```

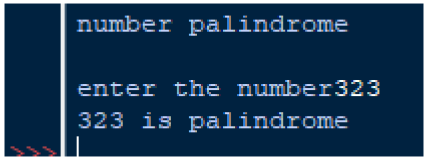


```
armstrong number

enter the number407
is armstrong
>>>
```

- Number palindrome or Not

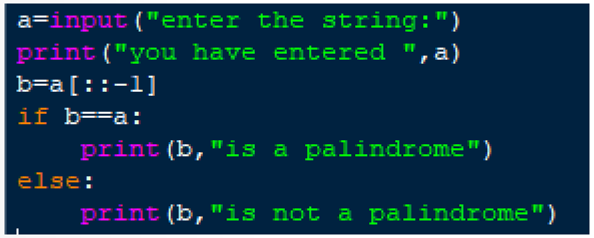


```
print("number palindrome")
print()
a=int(input("enter the number"))
rev=0
num=a
rev=0
for i in range(a):
    if a>0:
        b=a%10
        rev=rev*10+b
        #print(rev)
        d=a//10
        a=d
        i=i+1

if num==rev:
    print(rev,"is palindrome")

else:
    print("is not palindrome")
```

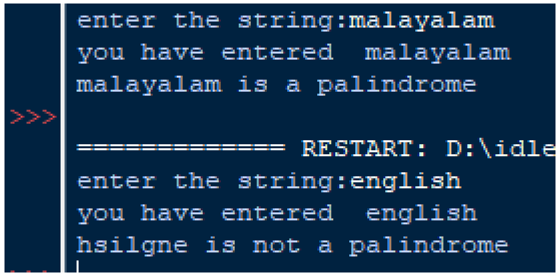


```
number palindrome

enter the number323
323 is palindrome
>>>
```

- String palindrome or Not

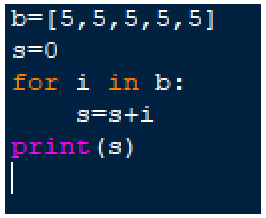


```
a=input("enter the string:")
print("you have entered ",a)
b=a[::-1]
if b==a:
    print(b,"is a palindrome")
else:
    print(b,"is not a palindrome")
```

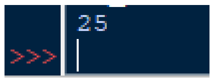


```
    enter the string:malayalam
    you have entered  malayalam
    malayalam is a palindrome
>>>
    ============= RESTART: D:\idle
    enter the string:english
    you have entered  english
    hsilgne is not a palindrome
```

- Sum of numbers in list

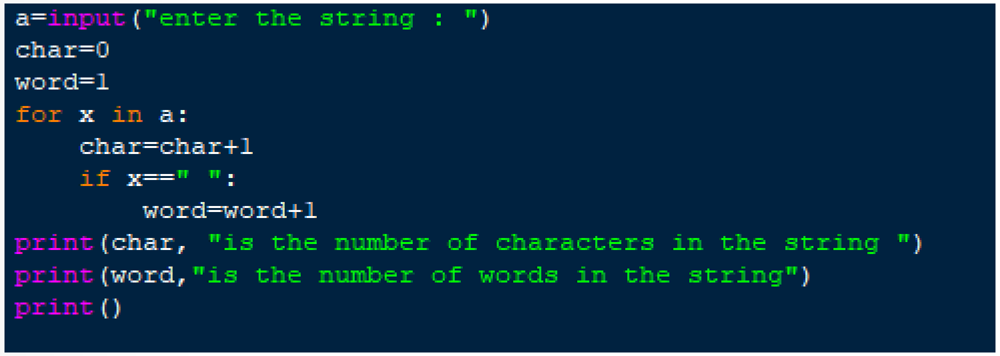


```
b=[5,5,5,5,5]
s=0
for i in b:
    s=s+i
print(s)
```

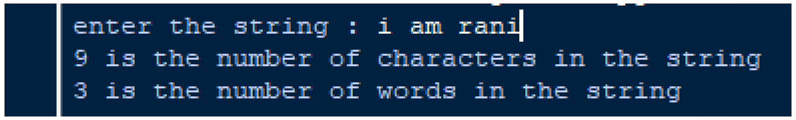


```
25
>>>
```

- Count the characters and word in a string



```python
a=input("enter the string : ")
char=0
word=1
for x in a:
    char=char+1
    if x==" ":
        word=word+1
print(char, "is the number of characters in the string ")
print(word,"is the number of words in the string")
print()
```



```
enter the string : i am rani
9 is the number of characters in the string
3 is the number of words in the string
```

While loop

- With the while loop we can execute a set of statements as long as a condition is true.

- We generally use this loop when we don't know the number of times to iterate beforehand.

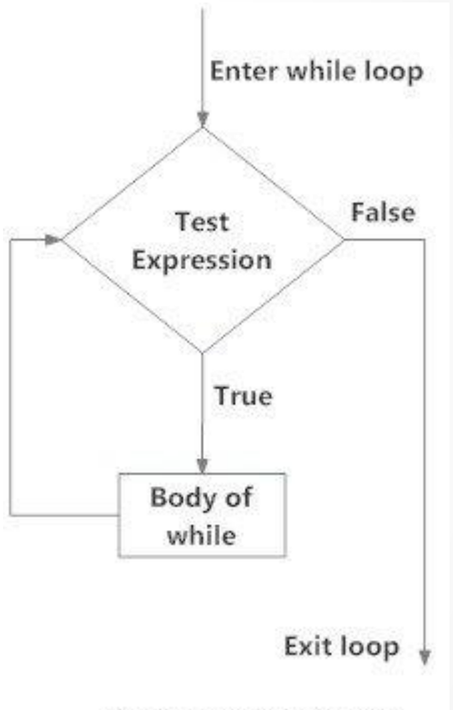
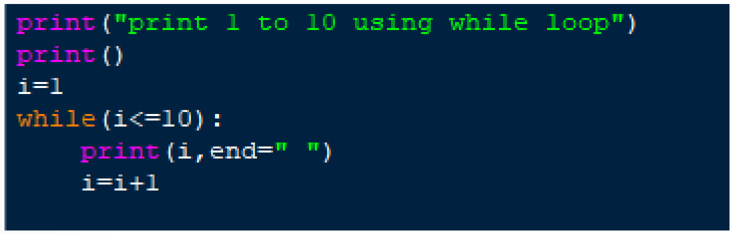
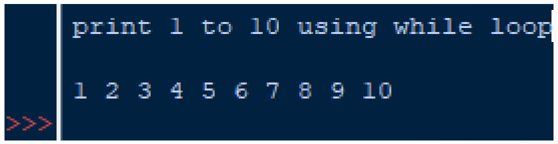


Fig: operation of while loop

Print 1 – 10 using while loop

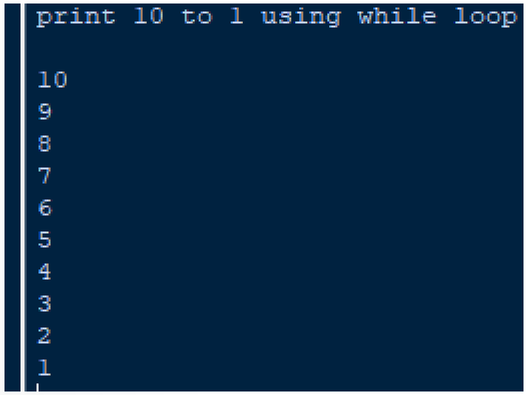


```
print("print 1 to 10 using while loop")
print()
i=1
while(i<=10):
    print(i,end=" ")
    i=i+1
```

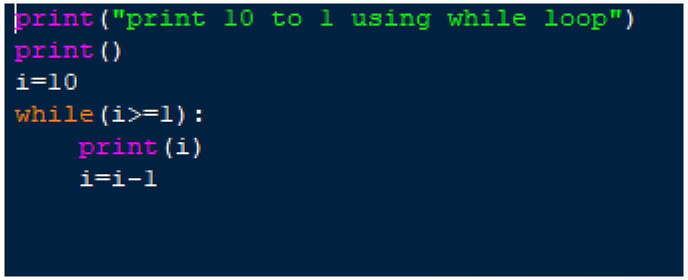


```
print 1 to 10 using while loop

1 2 3 4 5 6 7 8 9 10
>>>
```

Print 10 – 1 using while loop

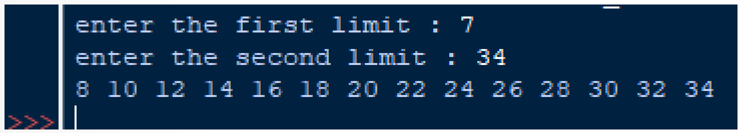


```
print 10 to 1 using while loop

10
9
8
7
6
5
4
3
2
1
```

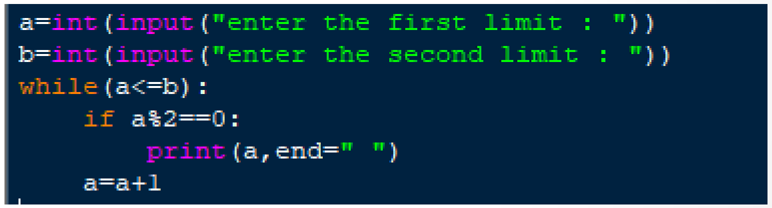


```
print("print 10 to 1 using while loop")
print()
i=10
while(i>=1):
    print(i)
    i=i-1
```

Even number in a limit (while loop)

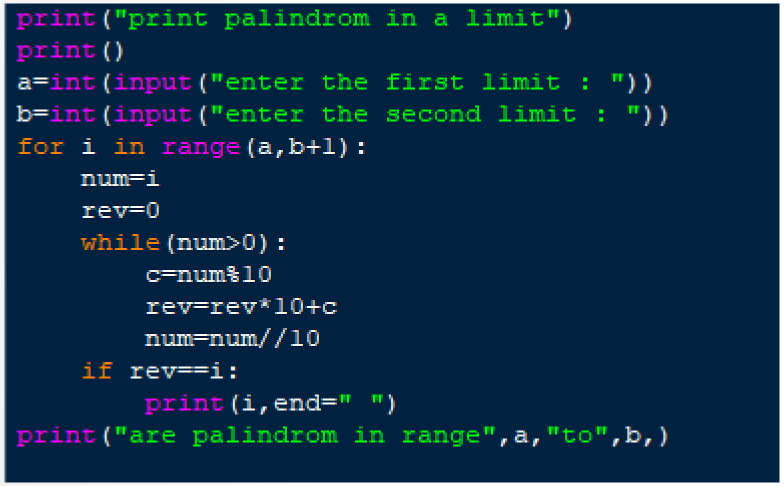


```
enter the first limit : 7
enter the second limit : 34
8 10 12 14 16 18 20 22 24 26 28 30 32 34
>>>
```

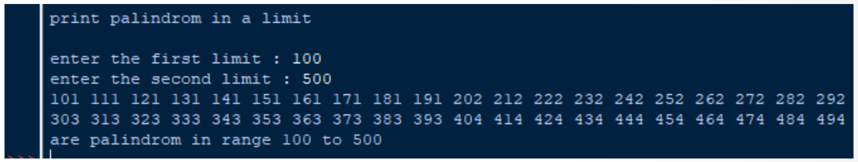


```
a=int(input("enter the first limit : "))
b=int(input("enter the second limit : "))
while(a<=b):
    if a%2==0:
        print(a,end=" ")
    a=a+1
```

Number palindrome in a limit(while loop)

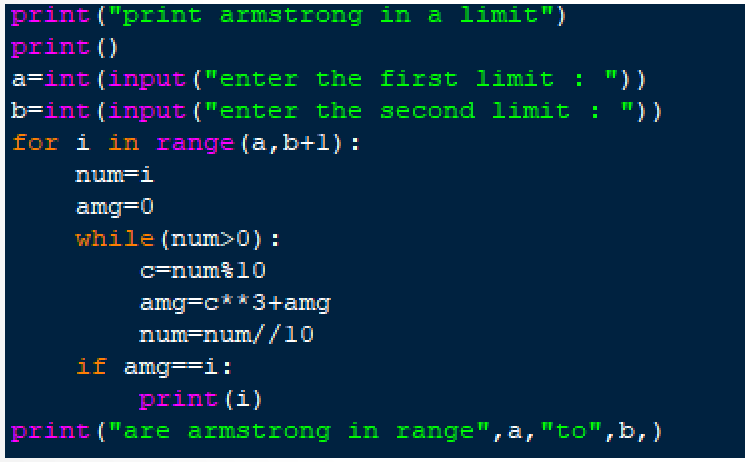


```
print("print palindrom in a limit")
print()
a=int(input("enter the first limit : "))
b=int(input("enter the second limit : "))
for i in range(a,b+1):
    num=i
    rev=0
    while(num>0):
        c=num%10
        rev=rev*10+c
        num=num//10
    if rev==i:
        print(i,end=" ")
print("are palindrom in range",a,"to",b,)
```

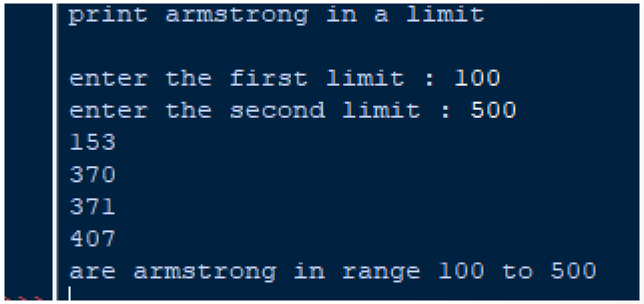


```
print palindrom in a limit

enter the first limit : 100
enter the second limit : 500
101 111 121 131 141 151 161 171 181 191 202 212 222 232 242 252 262 272 282 292
303 313 323 333 343 353 363 373 383 393 404 414 424 434 444 454 464 474 484 494
are palindrom in range 100 to 500
```

Armstrong number in a limit (while loop)



```
print("print armstrong in a limit")
print()
a=int(input("enter the first limit : "))
b=int(input("enter the second limit : "))
for i in range(a,b+1):
    num=i
    amg=0
    while(num>0):
        c=num%10
        amg=c**3+amg
        num=num//10
    if amg==i:
        print(i)
print("are armstrong in range",a,"to",b,)
```



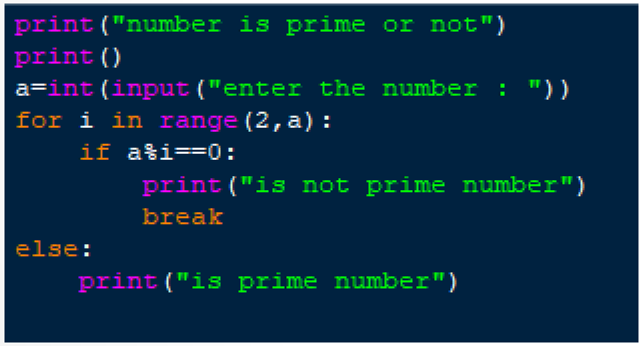
```
print armstrong in a limit

enter the first limit : 100
enter the second limit : 500
153
370
371
407
are armstrong in range 100 to 500
```

BREAK STATEMENTS:

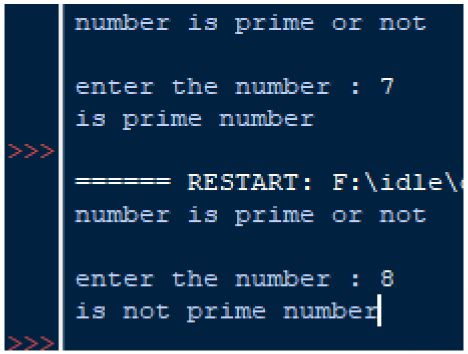
With the break statement we can stop the loop before it has looped through all the items:

```
#break statement
for x in range(6):
    if x == 3:
        break
    print(x)
else:
    print("finally finished!")
```

- A number prime or not



```
print("number is prime or not")
print()
a=int(input("enter the number : "))
for i in range(2,a):
    if a%i==0:
        print("is not prime number")
        break
else:
    print("is prime number")
```



```
    number is prime or not

    enter the number : 7
    is prime number
>>>
    ====== RESTART: F:\idle\
    number is prime or not

    enter the number : 8
    is not prime number
>>>
```

CONTINUE STATEMENT

- With the continue statement we can stop the current iteration of the loop, and continue with the next:

```
#continue statement
fruits = ["apple", "banana",
"cherry"]
for x in fruits:
   if x =="banana":
      continue
   print(x)
```

PYTHON FUCTION

Functions

- A function is a block of code that can only runs when it is called.
- You can pass data, known as parameters, into a function.
- Function can break your large program to small parts to do an specific job of the program.

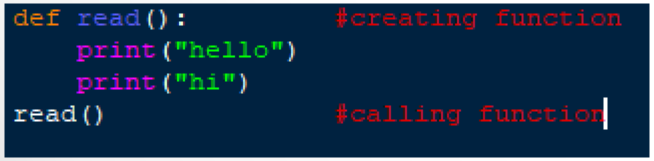
Creating a Function

In Python a function is defined using the **def** keyword:

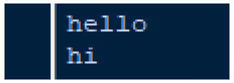
Calling a Function:

To call a function, use the function name followed by parenthesis

fun_name()



```
def read():             #creating function
    print("hello")
    print("hi")
read()                  #calling function
```



```
hello
hi
```

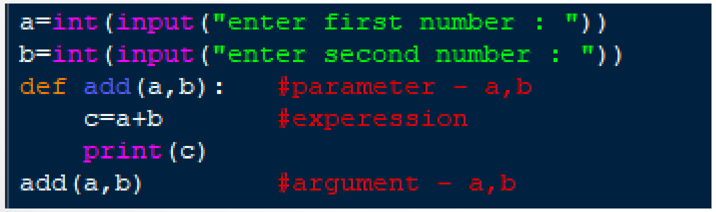
Arguments and parameters in functions

Parameters:

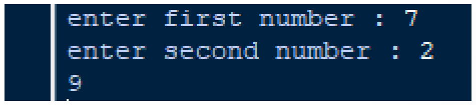
- parameters are the variable names within the function definition.

Arguments:

- Arguments are the values that passed in the function that is called

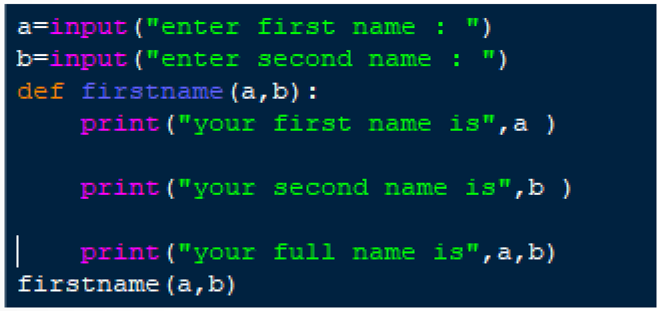


```python
a=int(input("enter first number : "))
b=int(input("enter second number : "))
def add(a,b):    #parameter - a,b
    c=a+b        #experession
    print(c)
add(a,b)         #argument - a,b
```

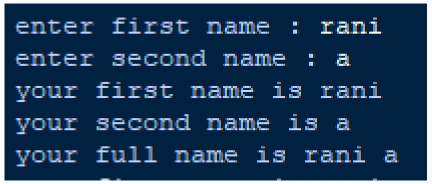


```
enter first number : 7
enter second number : 2
9
```

- we can use n (infinitive) number of parameter , argument ,expression in def function()

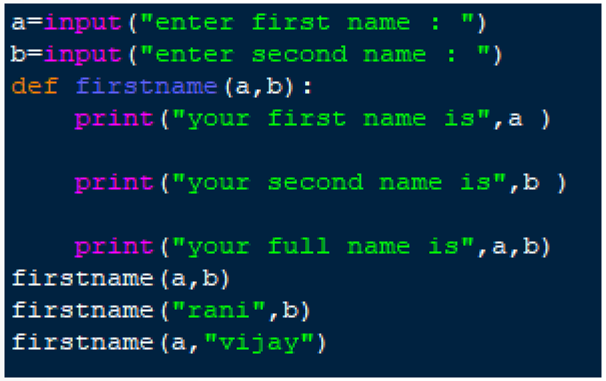


```
a=input("enter first name : ")
b=input("enter second name : ")
def firstname(a,b):
    print("your first name is",a )

    print("your second name is",b )

    print("your full name is",a,b)
firstname(a,b)
```

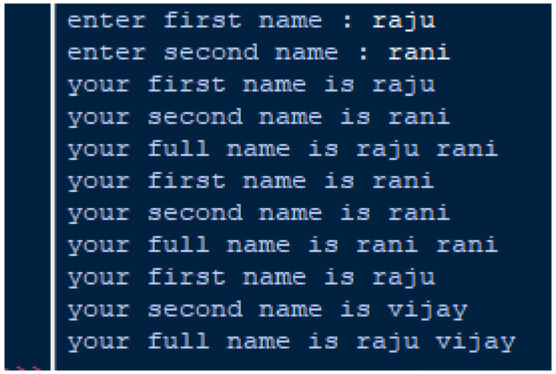


```
enter first name : rani
enter second name : a
your first name is rani
your second name is a
your full name is rani a
```

Multiple function call

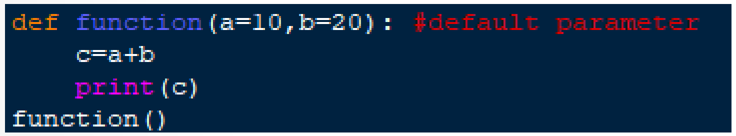


```
a=input("enter first name : ")
b=input("enter second name : ")
def firstname(a,b):
    print("your first name is",a )

    print("your second name is",b )

    print("your full name is",a,b)
firstname(a,b)
firstname("rani",b)
firstname(a,"vijay")
```

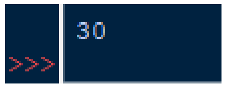


```
enter first name : raju
enter second name : rani
your first name is raju
your second name is rani
your full name is raju rani
your first name is rani
your second name is rani
your full name is rani rani
your first name is raju
your second name is vijay
your full name is raju vijay
```

Default parameter

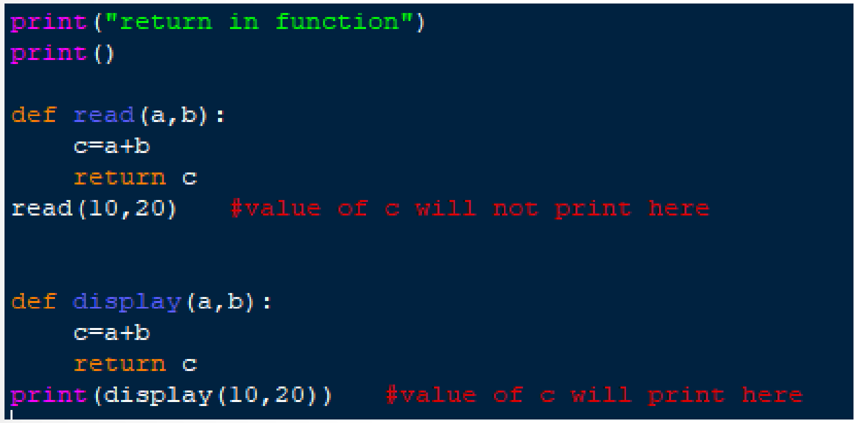


```
def function(a=10,b=20): #default parameter
    c=a+b
    print(c)
function()
```

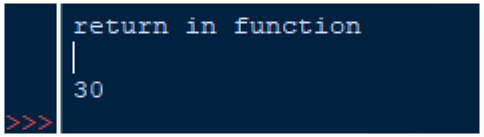


```
30
>>>
```

Return values in function



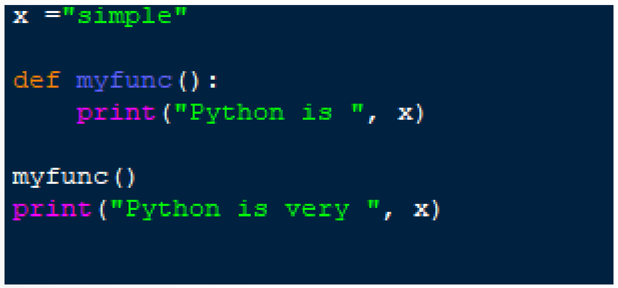
```
print("return in function")
print()

def read(a,b):
    c=a+b
    return c
read(10,20)    #value of c will not print here


def display(a,b):
    c=a+b
    return c
print(display(10,20))    #value of c will print here
```



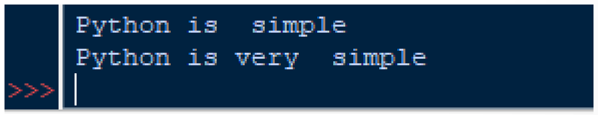
```
return in function

30
>>>
```

Global variable

- Variables that are created outside of a function are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.



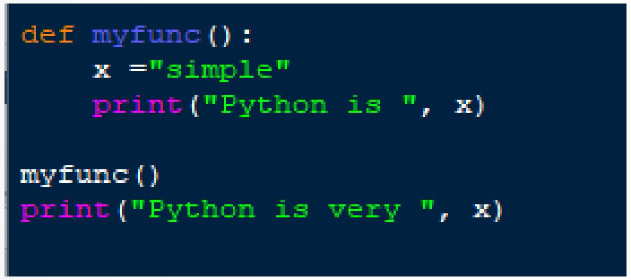
```
x ="simple"

def myfunc():
    print("Python is ", x)

myfunc()
print("Python is very ", x)
```



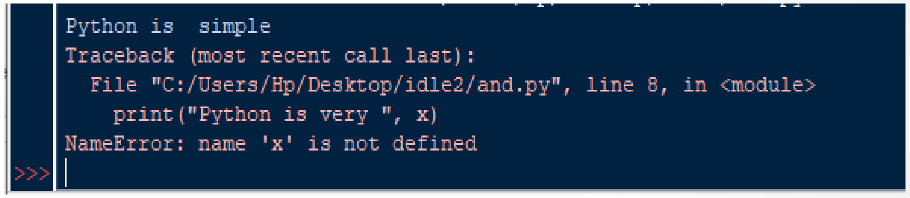
```
Python is  simple
Python is very  simple
>>>
```

Local variable

Local variable is declared inside the function

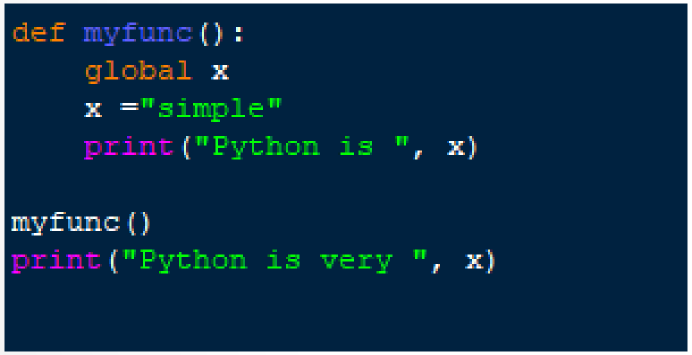


```
def myfunc():
    x ="simple"
    print("Python is ", x)

myfunc()
print("Python is very ", x)
```

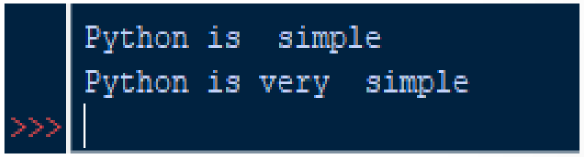


```
Python is  simple
Traceback (most recent call last):
  File "C:/Users/Hp/Desktop/idle2/and.py", line 8, in <module>
    print("Python is very ", x)
NameError: name 'x' is not defined
>>> 
```

Local variable to global variable



```
def myfunc():
    global x
    x ="simple"
    print("Python is ", x)

myfunc()
print("Python is very ", x)
```

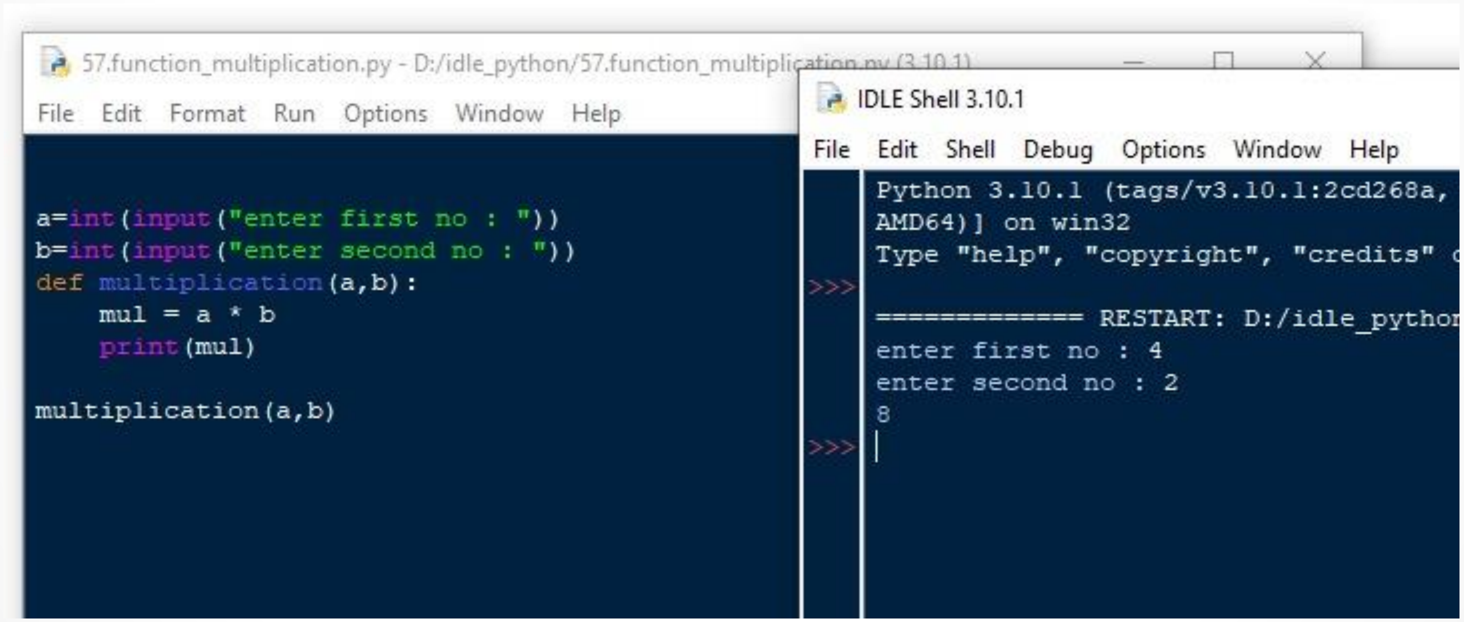


```
Python is  simple
Python is very  simple
>>>
```

Programs using function:

- Multiplication giving parameter
- Finding sum and average Of 2 Number
- Area and Perimeter of rectangle using function
- Calculator using function

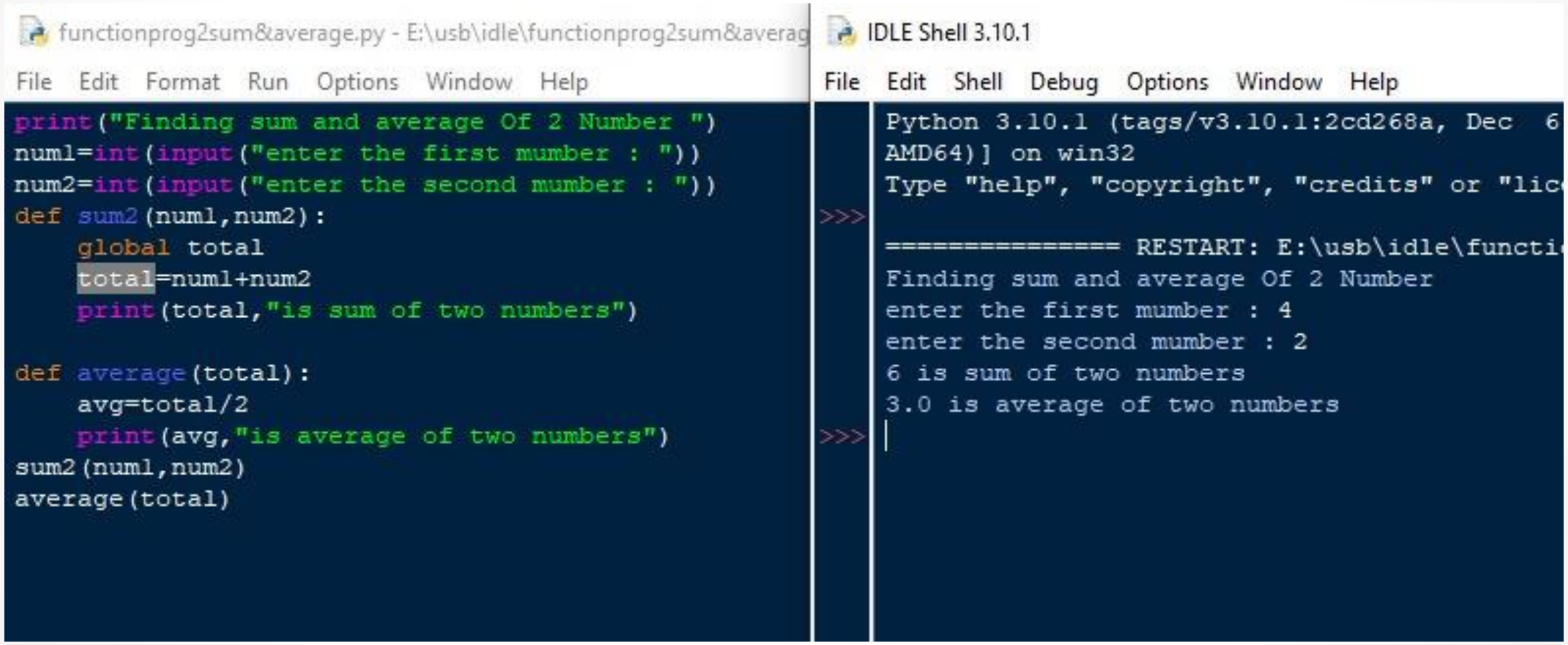
Multiplication giving parameter using function


57.function_multiplication.py - D:/idle_python/57.function_multiplication.py (3.10.1)
File Edit Format Run Options Window Help
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help

```
a=int(input("enter first no : "))
b=int(input("enter second no : "))
def multiplication(a,b):
    mul = a * b
    print(mul)

multiplication(a,b)
```

```
    Python 3.10.1 (tags/v3.10.1:2cd268a,
    AMD64)] on win32
    Type "help", "copyright", "credits" o
>>>
    ============= RESTART: D:/idle_pythor
    enter first no : 4
    enter second no : 2
    8
>>> |
```

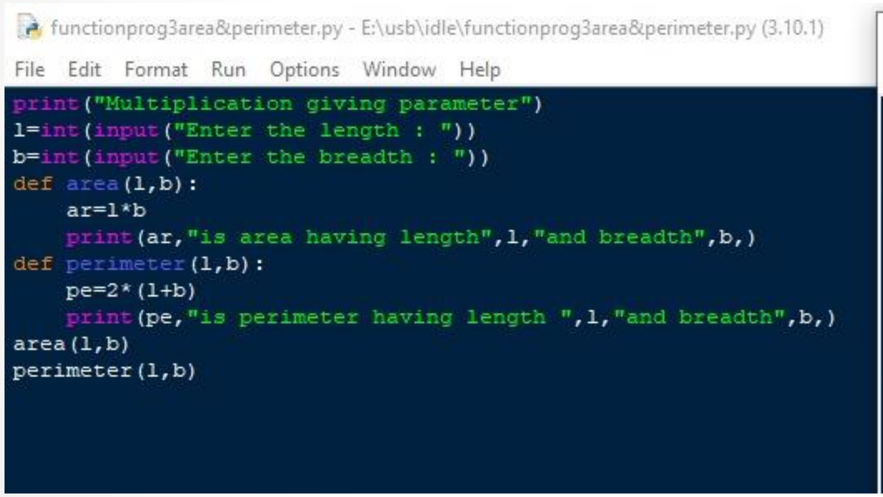
Finding sum and average Of 2 Number


functionprog2sum&average.py - E:\usb\idle\functionprog2sum&averag
File Edit Format Run Options Window Help
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help

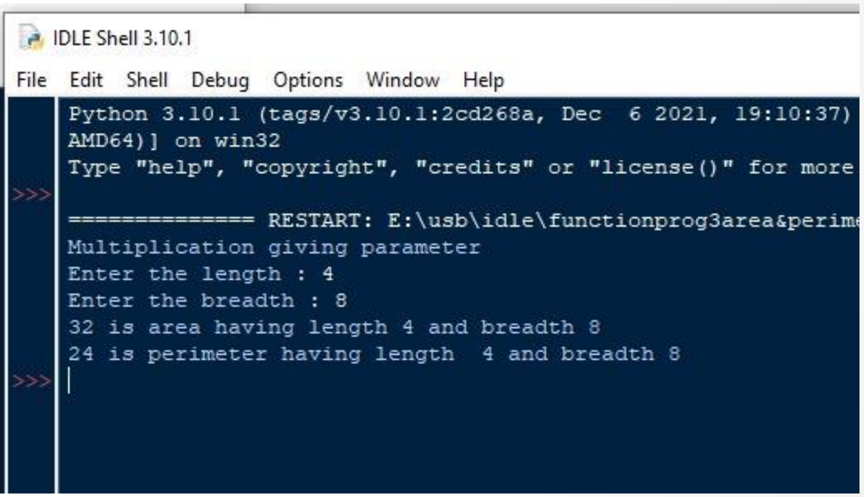
```
print("Finding sum and average Of 2 Number ")
num1=int(input("enter the first number : "))
num2=int(input("enter the second number : "))
def sum2(num1,num2):
    global total
    total=num1+num2
    print(total,"is sum of two numbers")

def average(total):
    avg=total/2
    print(avg,"is average of two numbers")
sum2(num1,num2)
average(total)
```

```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6
AMD64)] on win32
Type "help", "copyright", "credits" or "lic
>>>
=============== RESTART: E:\usb\idle\functi
Finding sum and average Of 2 Number
enter the first number : 4
enter the second number : 2
6 is sum of two numbers
3.0 is average of two numbers
>>>
```

Area and Perimeter of rectangle using function

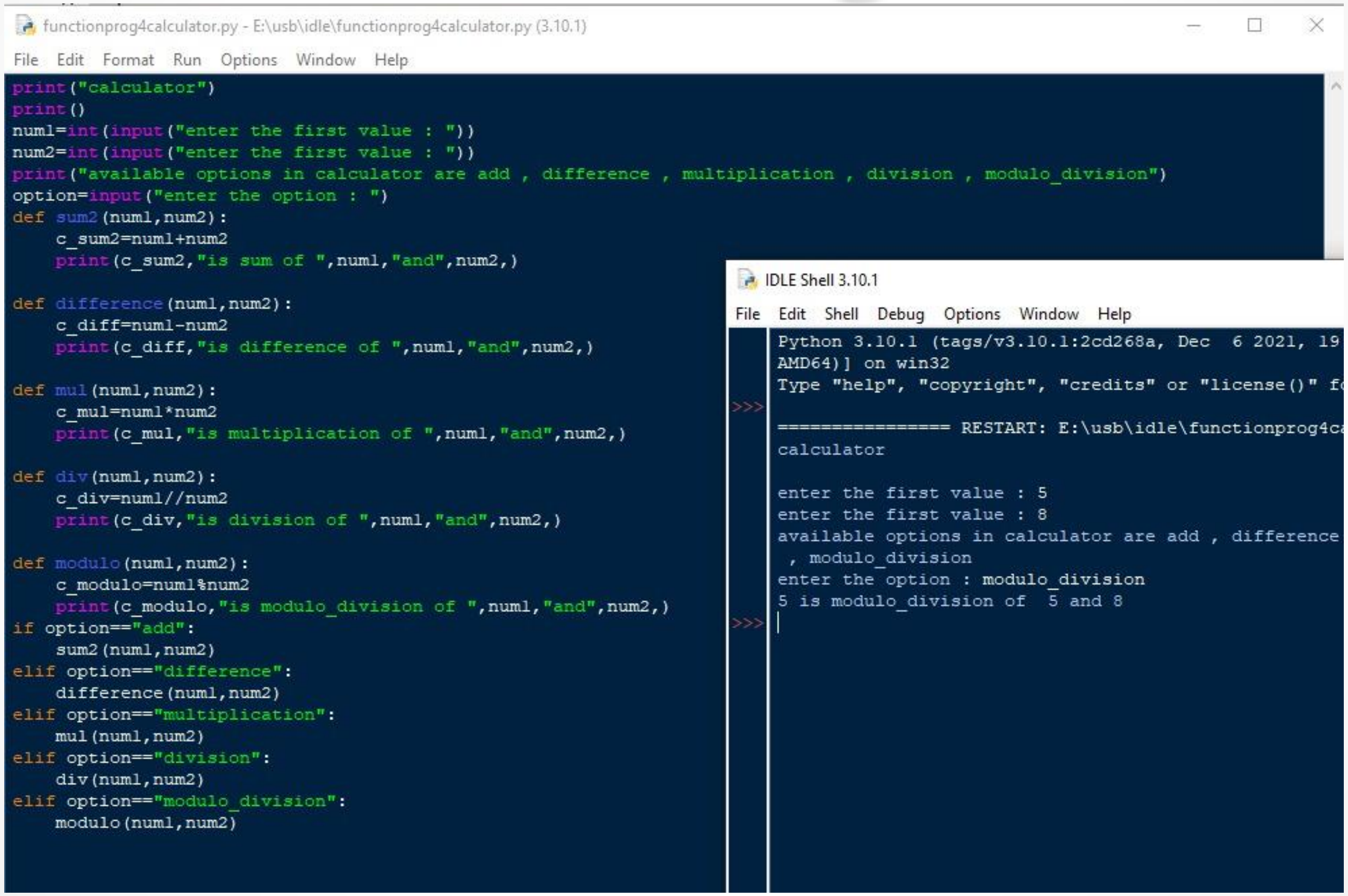

functionprog3area&perimeter.py - E:\usb\idle\functionprog3area&perimeter.py (3.10.1)

```python
print("Multiplication giving parameter")
l=int(input("Enter the length : "))
b=int(input("Enter the breadth : "))
def area(l,b):
    ar=l*b
    print(ar,"is area having length",l,"and breadth",b,)
def perimeter(l,b):
    pe=2*(l+b)
    print(pe,"is perimeter having length ",l,"and breadth",b,)
area(l,b)
perimeter(l,b)
```


IDLE Shell 3.10.1

```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37)
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
>>>
============== RESTART: E:\usb\idle\functionprog3area&perim
Multiplication giving parameter
Enter the length : 4
Enter the breadth : 8
32 is area having length 4 and breadth 8
24 is perimeter having length  4 and breadth 8
>>>
```

Calculator using function



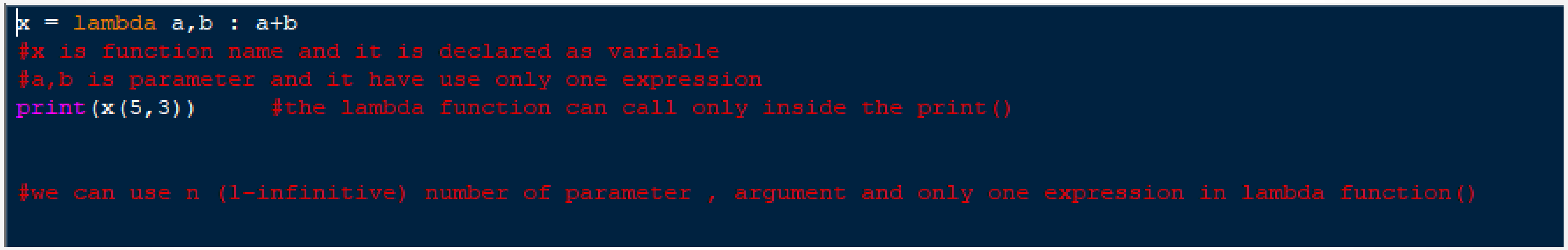
functionprog4calculator.py - E:\usb\idle\functionprog4calculator.py (3.10.1)
File Edit Format Run Options Window Help
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help

```python
print("calculator")
print()
num1=int(input("enter the first value : "))
num2=int(input("enter the first value : "))
print("available options in calculator are add , difference , multiplication , division , modulo_division")
option=input("enter the option : ")
def sum2(num1,num2):
    c_sum2=num1+num2
    print(c_sum2,"is sum of ",num1,"and",num2,)

def difference(num1,num2):
    c_diff=num1-num2
    print(c_diff,"is difference of ",num1,"and",num2,)

def mul(num1,num2):
    c_mul=num1*num2
    print(c_mul,"is multiplication of ",num1,"and",num2,)

def div(num1,num2):
    c_div=num1//num2
    print(c_div,"is division of ",num1,"and",num2,)

def modulo(num1,num2):
    c_modulo=num1%num2
    print(c_modulo,"is modulo_division of ",num1,"and",num2,)
if option=="add":
    sum2(num1,num2)
elif option=="difference":
    difference(num1,num2)
elif option=="multiplication":
    mul(num1,num2)
elif option=="division":
    div(num1,num2)
elif option=="modulo_division":
    modulo(num1,num2)
```

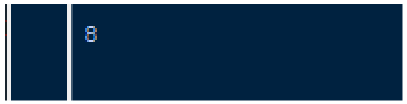
```
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
>>>
================ RESTART: E:\usb\idle\functionprog4ca
calculator

enter the first value : 5
enter the first value : 8
available options in calculator are add , difference
 , modulo_division
enter the option : modulo_division
5 is modulo_division of  5 and 8
>>>
```

Lambda function

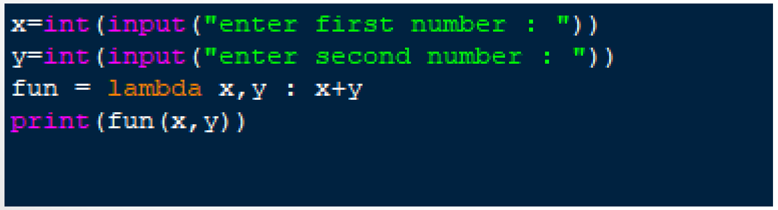
- A lambda function is an **anonymous** and **one-use-only** function that can have **any number of parameters** and that does some computation.



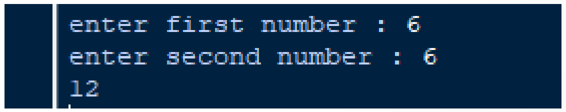
```python
x = lambda a,b : a+b
#x is function name and it is declared as variable
#a,b is parameter and it have use only one expression
print(x(5,3))      #the lambda function can call only inside the print()


#we can use n (1-infinitive) number of parameter , argument and only one expression in lambda function()
```



```
8
```



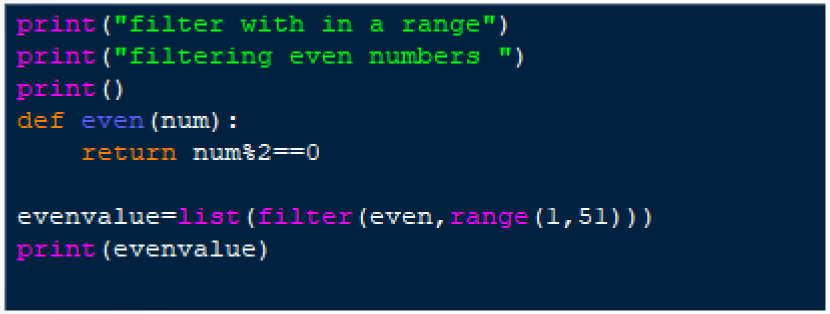
```python
x=int(input("enter first number : "))
y=int(input("enter second number : "))
fun = lambda x,y : x+y
print(fun(x,y))
```



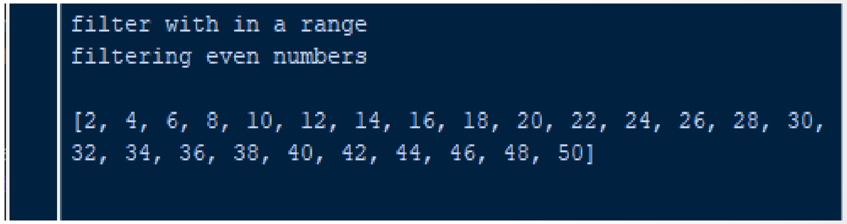
```
enter first number : 6
enter second number : 6
12
```

Filter function

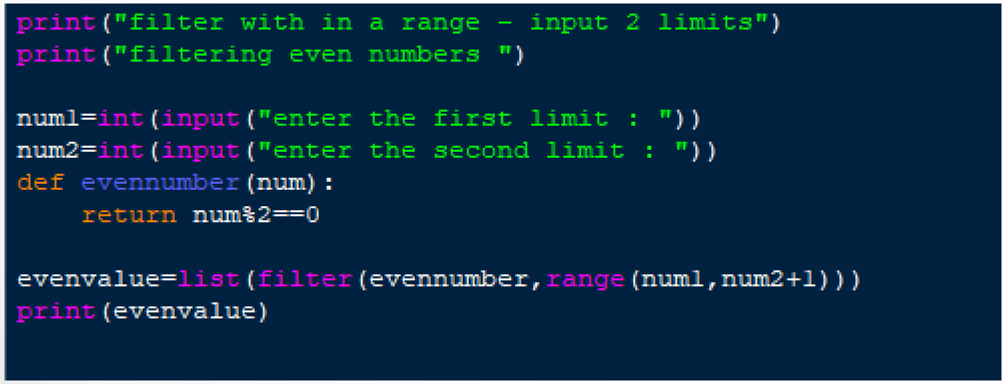
- The filter() method returns a sequence consisting of those elements for which the included function **returns true**, those that **satisfy the criteria** given in the specified function.



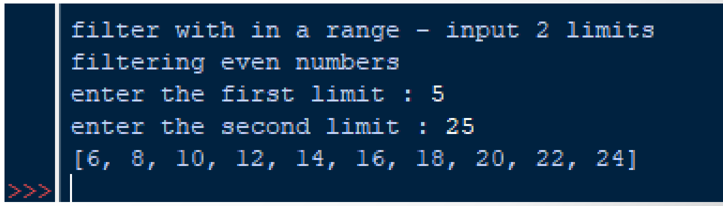
```
print("filter with in a range")
print("filtering even numbers ")
print()
def even(num):
    return num%2==0

evenvalue=list(filter(even,range(1,51)))
print(evenvalue)
```



```
filter with in a range
filtering even numbers

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
32, 34, 36, 38, 40, 42, 44, 46, 48, 50]
```

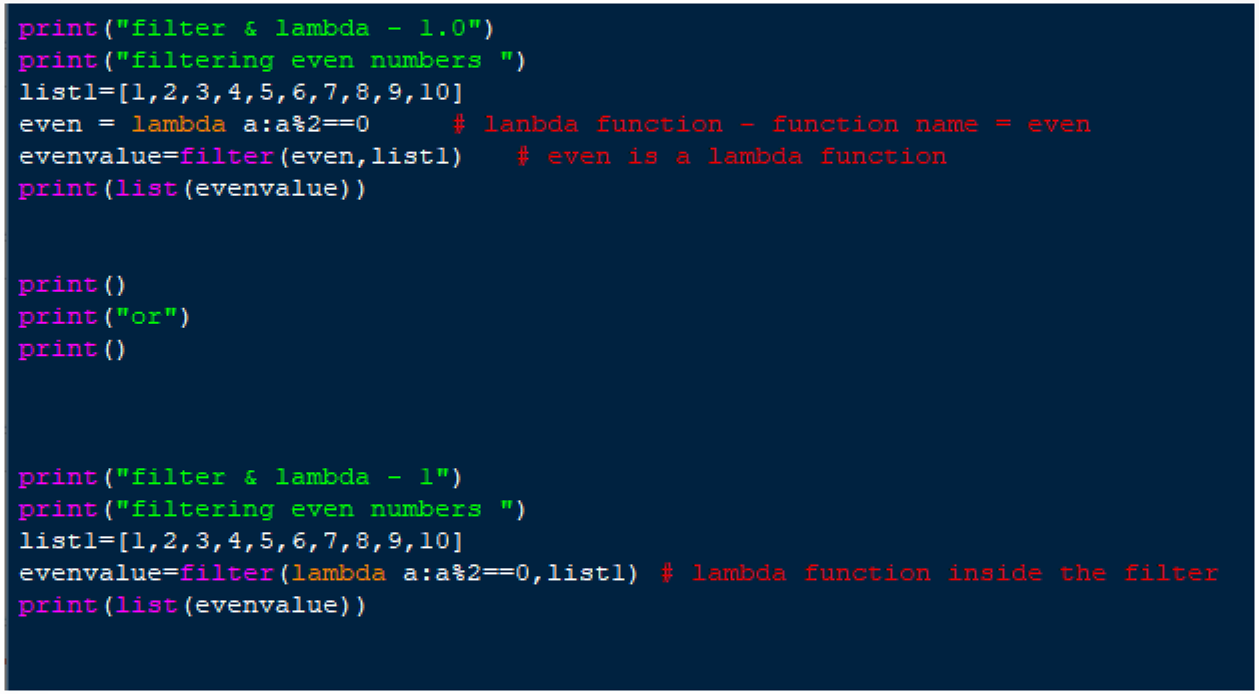


```
print("filter with in a range - input 2 limits")
print("filtering even numbers ")

num1=int(input("enter the first limit : "))
num2=int(input("enter the second limit : "))
def evennumber(num):
    return num%2==0

evenvalue=list(filter(evennumber,range(num1,num2+1)))
print(evenvalue)
```

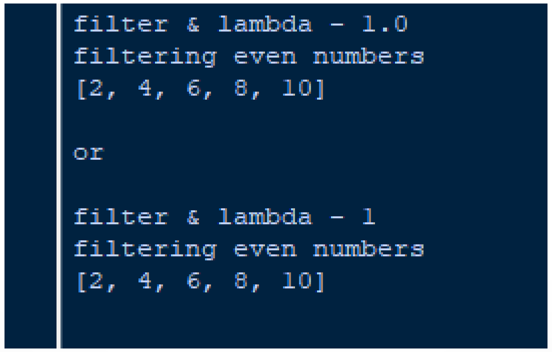


```
filter with in a range - input 2 limits
filtering even numbers
enter the first limit : 5
enter the second limit : 25
[6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
>>>
```

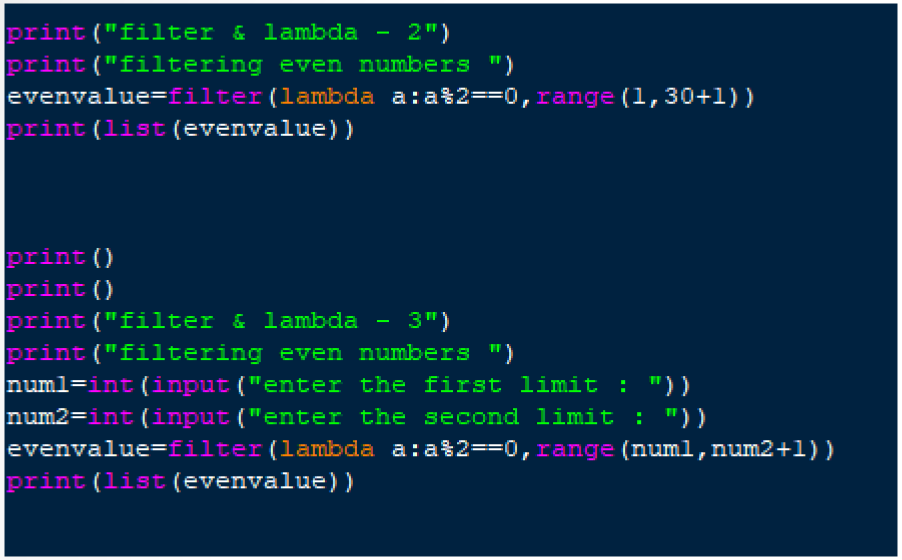
Filter & lambda



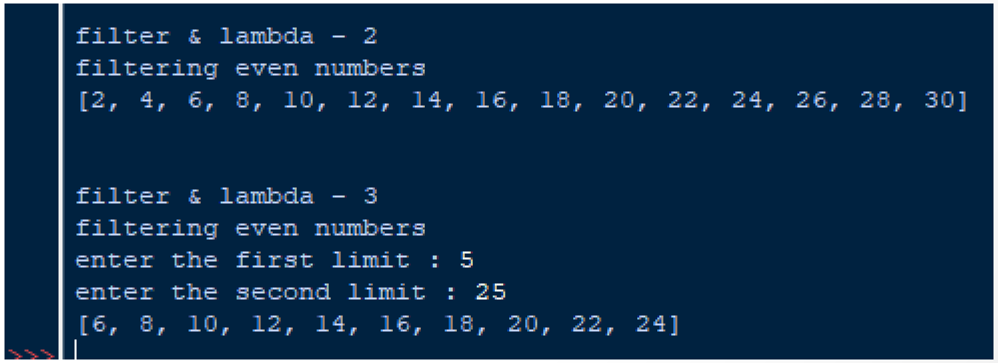
```
print("filter & lambda - 1.0")
print("filtering even numbers ")
listl=[1,2,3,4,5,6,7,8,9,10]
even = lambda a:a%2==0      # lanbda function - function name = even
evenvalue=filter(even,listl)    # even is a lambda function
print(list(evenvalue))


print()
print("or")
print()



print("filter & lambda - 1")
print("filtering even numbers ")
listl=[1,2,3,4,5,6,7,8,9,10]
evenvalue=filter(lambda a:a%2==0,listl) # lambda function inside the filter
print(list(evenvalue))
```



```
filter & lambda - 1.0
filtering even numbers
[2, 4, 6, 8, 10]

or

filter & lambda - 1
filtering even numbers
[2, 4, 6, 8, 10]
```



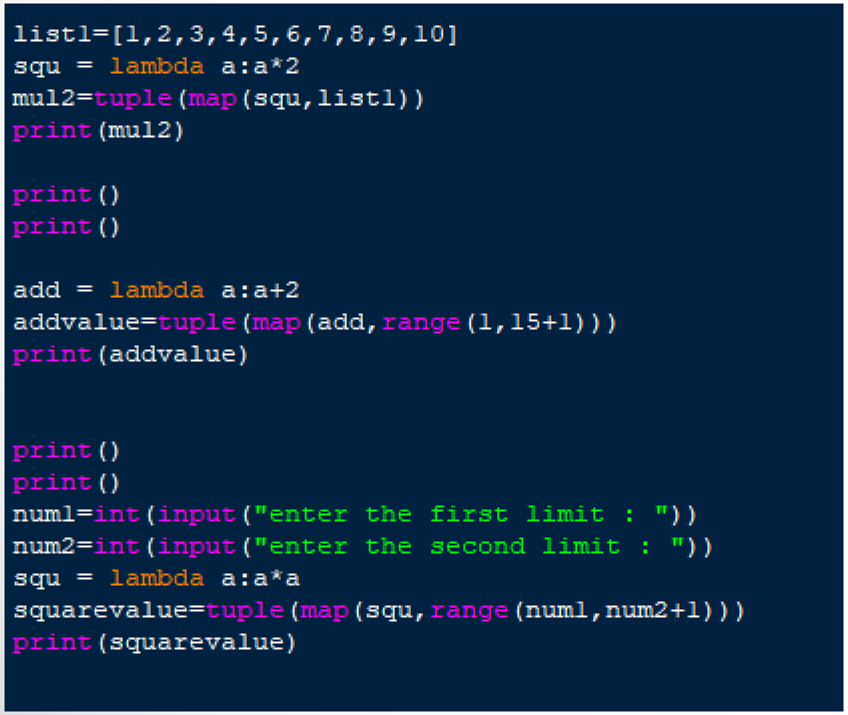
```python
print("filter & lambda - 2")
print("filtering even numbers ")
evenvalue=filter(lambda a:a%2==0,range(1,30+1))
print(list(evenvalue))



print()
print()
print("filter & lambda - 3")
print("filtering even numbers ")
num1=int(input("enter the first limit : "))
num2=int(input("enter the second limit : "))
evenvalue=filter(lambda a:a%2==0,range(num1,num2+1))
print(list(evenvalue))
```



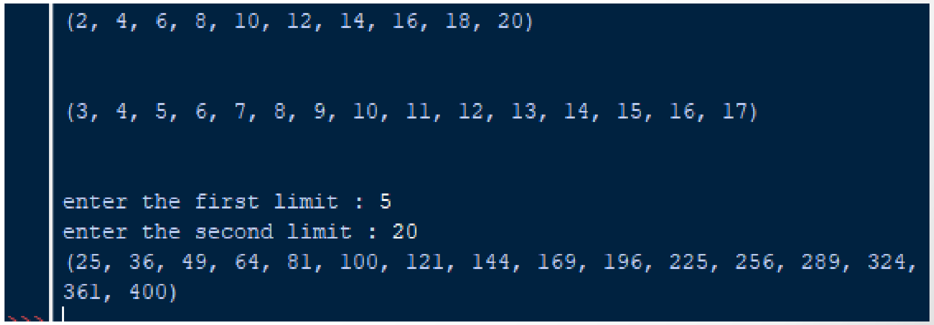
```
filter & lambda - 2
filtering even numbers
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]


filter & lambda - 3
filtering even numbers
enter the first limit : 5
enter the second limit : 25
[6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
>>> |
```

Map function

- The map method calls the included function for **each of the elements** in the sequence and **returns a list** of the returned values.



```
list1=[1,2,3,4,5,6,7,8,9,10]
squ = lambda a:a*2
mul2=tuple(map(squ,list1))
print(mul2)

print()
print()

add = lambda a:a+2
addvalue=tuple(map(add,range(1,15+1)))
print(addvalue)


print()
print()
num1=int(input("enter the first limit : "))
num2=int(input("enter the second limit : "))
squ = lambda a:a*a
squarevalue=tuple(map(squ,range(num1,num2+1)))
print(squarevalue)
```



```
(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)


(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17)


enter the first limit : 5
enter the second limit : 20
(25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324,
361, 400)
```

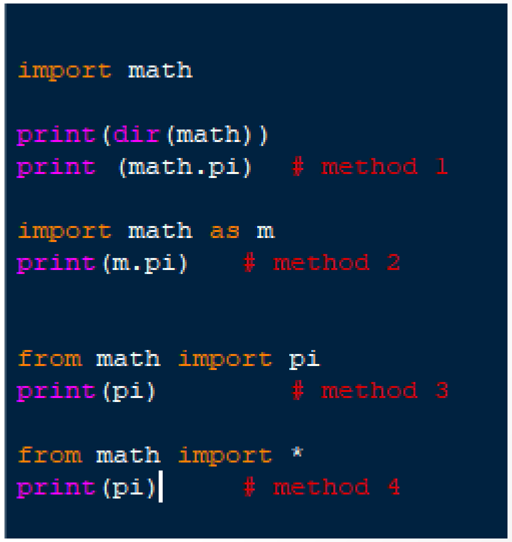
Modules

- A module is **a file consisting of Python code**.
- It can define **functions, classes, and variables**, and can also include **runnable code**.
- Any Python file can be referenced as a module.
- A file containing Python code, for example: test.py , is called a module, and its name would be test .

IMPORTING MODULES

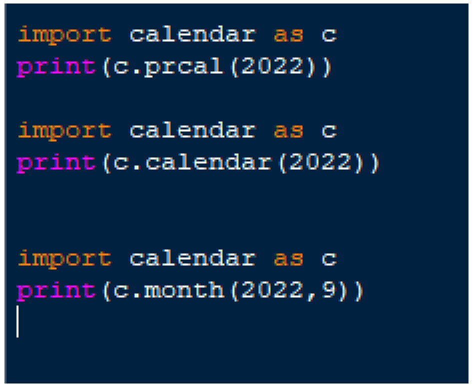
DIFFERENT WAYS TO IMPORT A MODULE

- Import module
- From module import function
- From module import *
- Import module as name

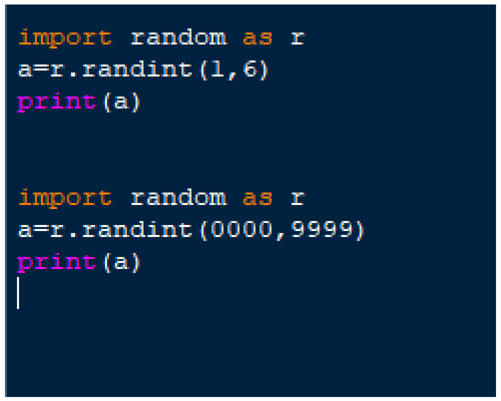


```
import math

print(dir(math))
print (math.pi)   # method 1

import math as m
print(m.pi)    # method 2


from math import pi
print(pi)          # method 3

from math import *
print(pi)      # method 4
```

Consider a calendar module that displays a calendar of a specified month and year

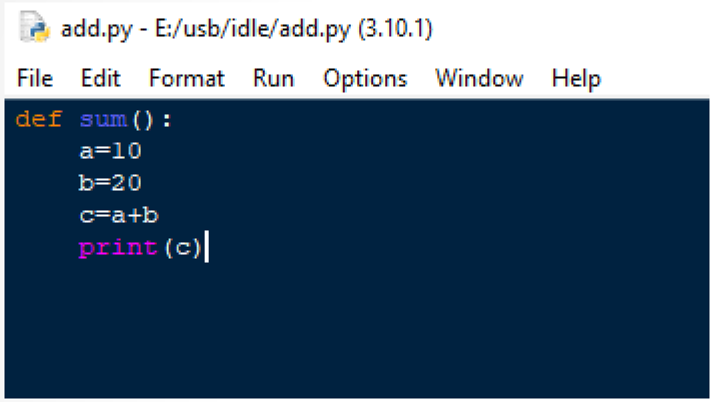


```
import calendar as c
print(c.prcal(2022))

import calendar as c
print(c.calendar(2022))


import calendar as c
print(c.month(2022,9))
```

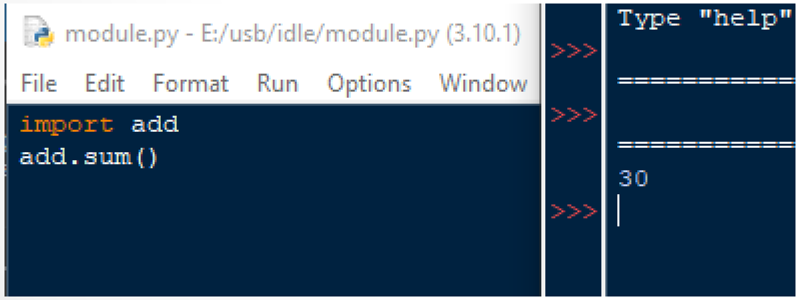


```
import random as r
a=r.randint(1,6)
print(a)


import random as r
a=r.randint(0000,9999)
print(a)
```

Importing a file


add.py - E:/usb/idle/add.py (3.10.1)
File Edit Format Run Options Window Help

```
def sum():
    a=10
    b=20
    c=a+b
    print(c)
```


module.py - E:/usb/idle/module.py (3.10.1)
File Edit Format Run Options Window

```
import add
add.sum()
```

```
    Type "help"
>>>
    ===========
>>>
    ===========
    30
>>>
```

Python oops concept

Python OOPs Concepts :

- Python is an object-oriented programming language.
- It allows us to develop applications using Object Oriented approach.
- In Python, we can easily create and use classes and objects.

- Programming based on style on object

Major principles of object-oriented programming system

- Object
- Class
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

Class & objects

- **Class**
- Blueprint of object
- To Create a class , use the keyword class
- **Object**
- Member of a class
- Object has properties and methods

- **Example**
- Smartphone , laptop , tv ->electronics

  Electronics-class

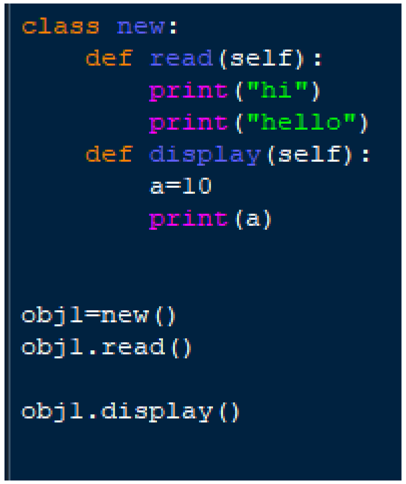
  Smartphone , laptop , tv-object
- Car , auto , bus -> vehicles

- **Example for creating class:**
- **Syntax**
- class class_name:
- To create a class name ‘newclass’ , with a property named x :

```
class newclass :
x = 5
```

- **Example for creating object :**
- **Syntax**
- Objectname = classname()
- Create an object named obj, and print the value of x :

```
obj = newclass()
print(obj.x)
```



```
class new:
    def read(self):
        print("hi")
        print("hello")
    def display(self):
        a=10
        print(a)


obj1=new()
obj1.read()

obj1.display()
```

What is ‘self’ !

- The self parameter is a reference to the class itself , and is used to access variables that belongs to the class
- It does not have to be named self , you can call it whatever you like , But it has to be the first parameter of any class

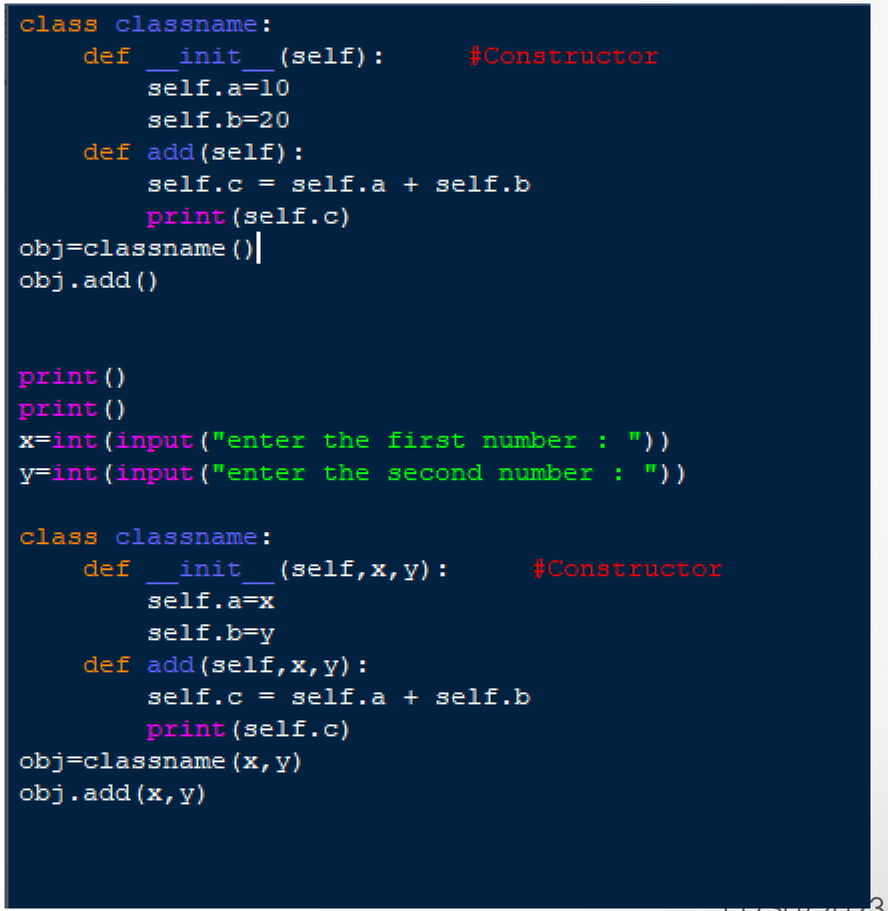
Syntax :

```
            def read (self):
```

Constructor __init__() Method

- Constructor is used for access the class directly

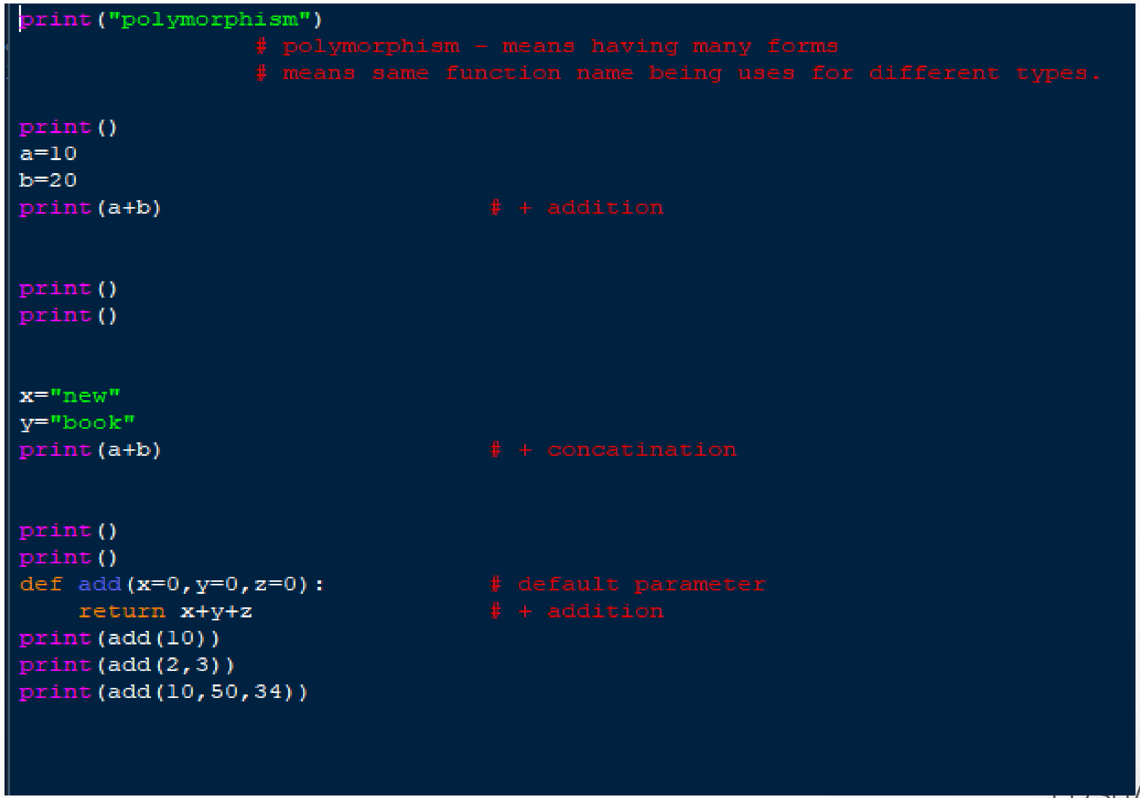
```
class rect:
        def __init__(self):
                self.l = 8
                self.b = 5

r=rect()
```



```
class classname:
    def __init__(self):      #Constructor
        self.a=10
        self.b=20
    def add(self):
        self.c = self.a + self.b
        print(self.c)
obj=classname()
obj.add()


print()
print()
x=int(input("enter the first number : "))
y=int(input("enter the second number : "))

class classname:
    def __init__(self,x,y):      #Constructor
        self.a=x
        self.b=y
    def add(self,x,y):
        self.c = self.a + self.b
        print(self.c)
obj=classname(x,y)
obj.add(x,y)
```

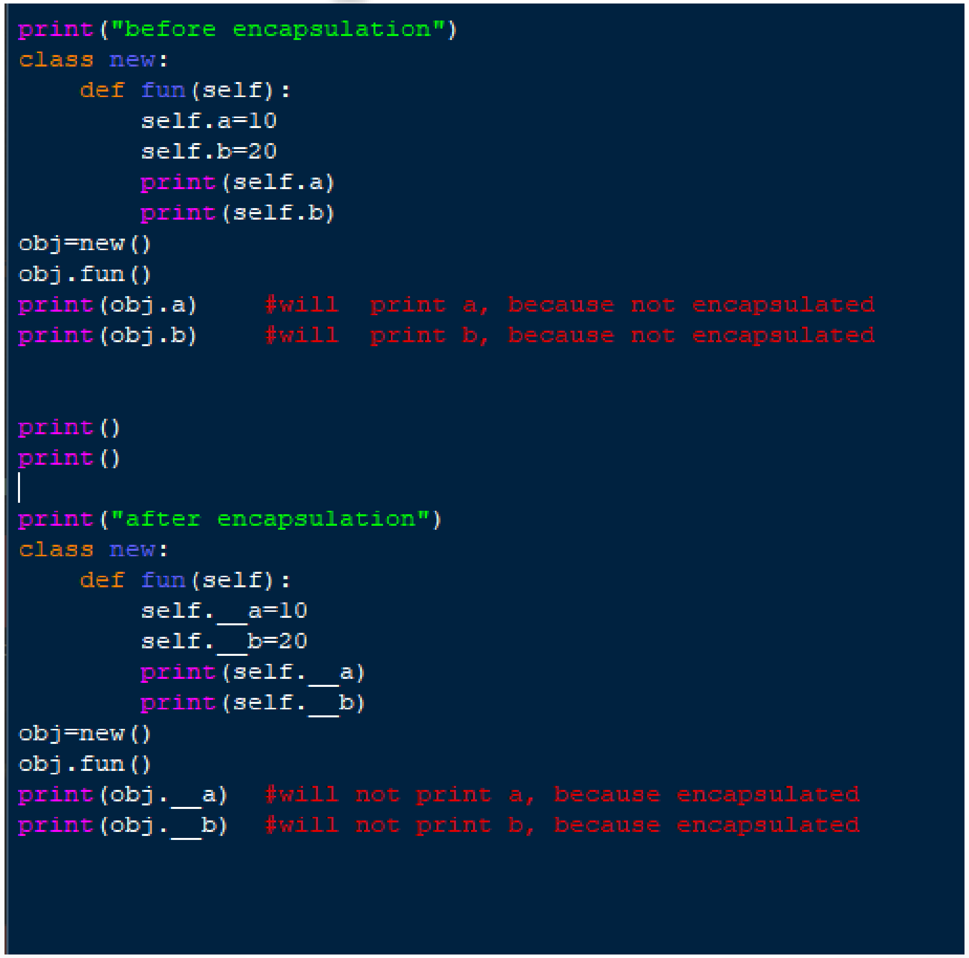
Polymorphism in python

- The word polymorphism means having many forms.
- In programming, polymorphism means same function name (but different signatures) being uses for different types.



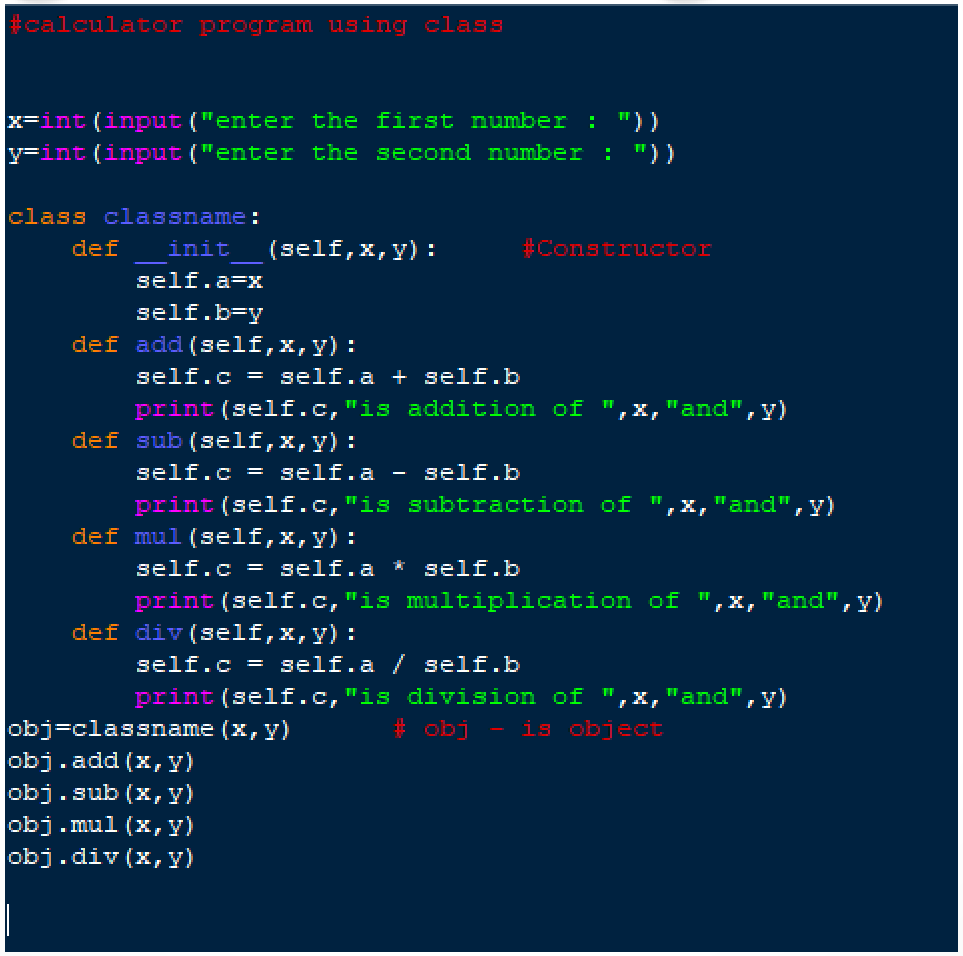
```
print("polymorphism")
                    # polymorphism - means having many forms
                    # means same function name being uses for different types.

print()
a=10
b=20
print(a+b)                              # + addition


print()
print()


x="new"
y="book"
print(a+b)                              # + concatination


print()
print()
def add(x=0,y=0,z=0):                   # default parameter
     return x+y+z                       # + addition
print(add(10))
print(add(2,3))
print(add(10,50,34))
```

encapsulation



```
print("before encapsulation")
class new:
    def fun(self):
        self.a=10
        self.b=20
        print(self.a)
        print(self.b)
obj=new()
obj.fun()
print(obj.a)    #will  print a, because not encapsulated
print(obj.b)    #will  print b, because not encapsulated


print()
print()

print("after encapsulation")
class new:
    def fun(self):
        self.__a=10
        self.__b=20
        print(self.__a)
        print(self.__b)
obj=new()
obj.fun()
print(obj.__a)  #will not print a, because encapsulated
print(obj.__b)  #will not print b, because encapsulated
```

Write a calculator program using class ?



```
#calculator program using class


x=int(input("enter the first number : "))
y=int(input("enter the second number : "))

class classname:
    def __init__(self,x,y):      #Constructor
        self.a=x
        self.b=y
    def add(self,x,y):
        self.c = self.a + self.b
        print(self.c,"is addition of ",x,"and",y)
    def sub(self,x,y):
        self.c = self.a - self.b
        print(self.c,"is subtraction of ",x,"and",y)
    def mul(self,x,y):
        self.c = self.a * self.b
        print(self.c,"is multiplication of ",x,"and",y)
    def div(self,x,y):
        self.c = self.a / self.b
        print(self.c,"is division of ",x,"and",y)
obj=classname(x,y)       # obj - is object
obj.add(x,y)
obj.sub(x,y)
obj.mul(x,y)
obj.div(x,y)
```

Class Inheritance

Super class

- super class is a class which is also called as the parent class or base class

super class is the class which has some sub classes

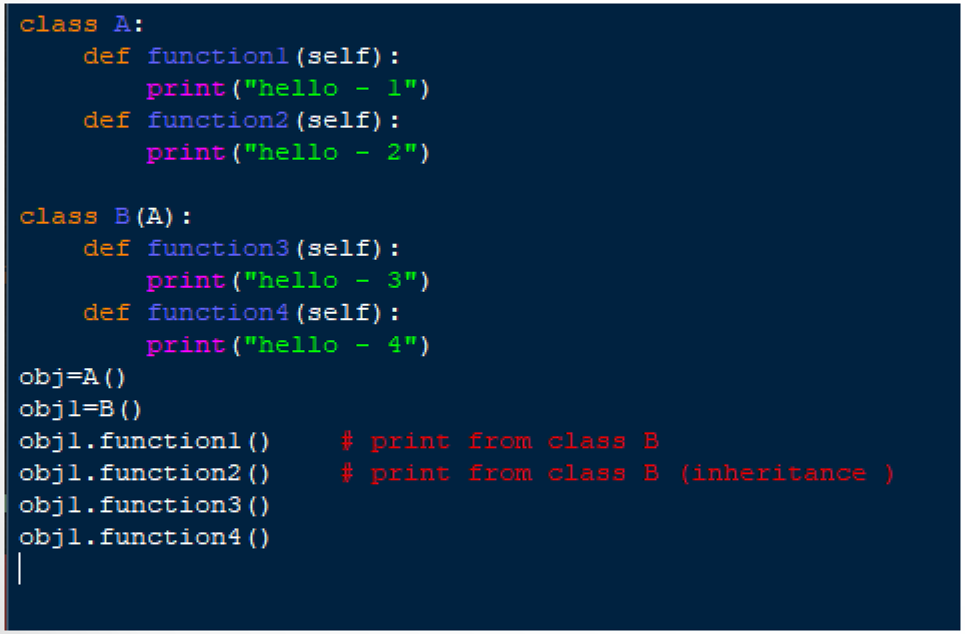
- There are multiple number of super class may or may not be available in a single program

Sub class

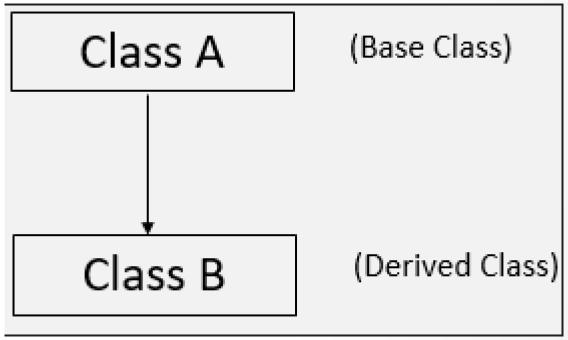
- Sub class is a class which is also called as the child class or derived class
- Sub class is the classes are derived from a base class
- Many sub class are derived from a single super class

Single inheritance

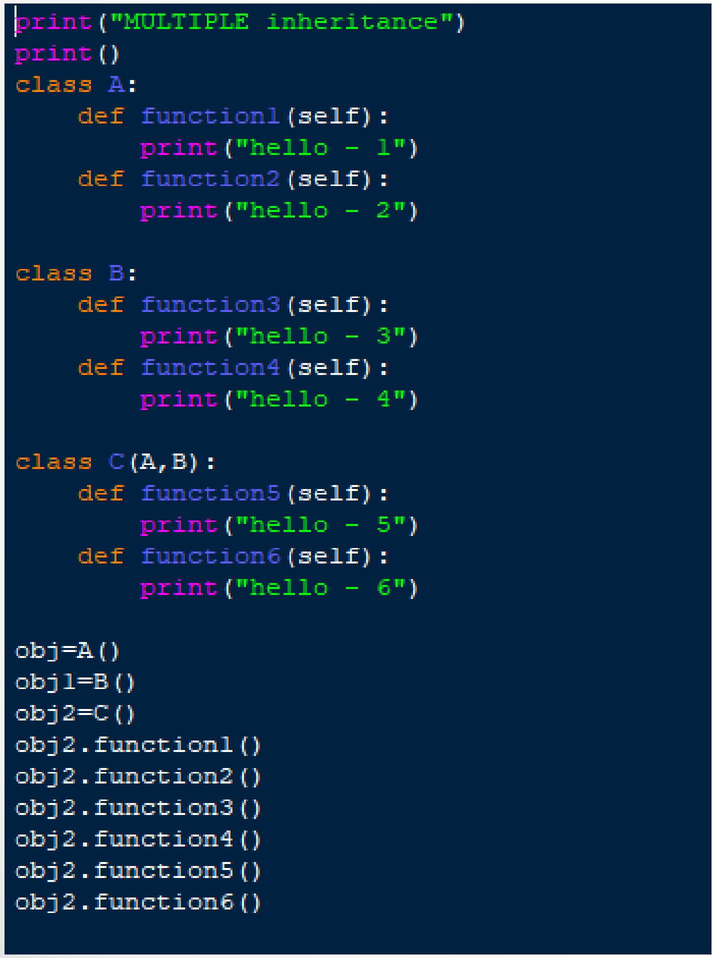
one class is derived from another single class



```
class A:
    def function1(self):
        print("hello - 1")
    def function2(self):
        print("hello - 2")

class B(A):
    def function3(self):
        print("hello - 3")
    def function4(self):
        print("hello - 4")
obj=A()
obj1=B()
obj1.function1()     # print from class B
obj1.function2()     # print from class B (inheritance )
obj1.function3()
obj1.function4()
```

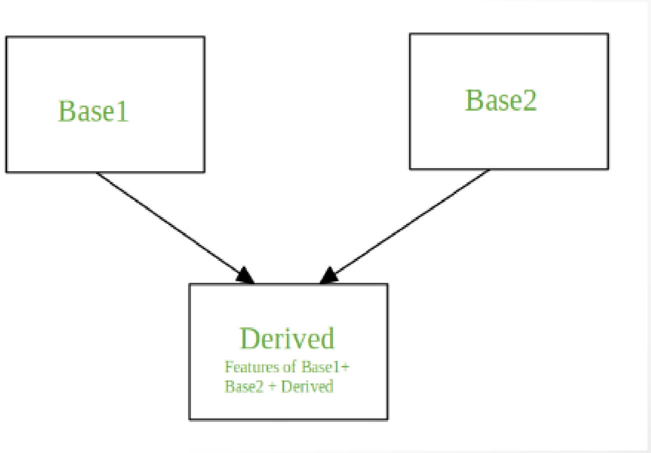


Multiple inheritance

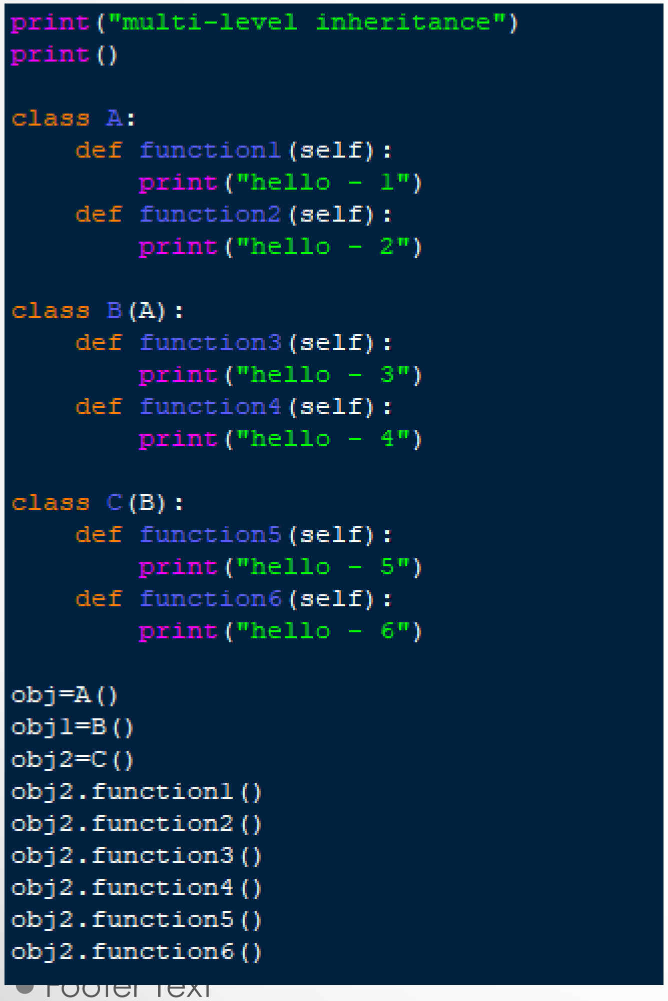


```
print("MULTIPLE inheritance")
print()
class A:
    def function1(self):
        print("hello - 1")
    def function2(self):
        print("hello - 2")

class B:
    def function3(self):
        print("hello - 3")
    def function4(self):
        print("hello - 4")

class C(A,B):
    def function5(self):
        print("hello - 5")
    def function6(self):
        print("hello - 6")

obj=A()
obj1=B()
obj2=C()
obj2.function1()
obj2.function2()
obj2.function3()
obj2.function4()
obj2.function5()
obj2.function6()
```

A class can be derived from more than one base class .

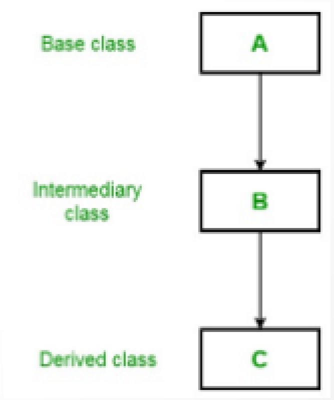


Multi level inheritance

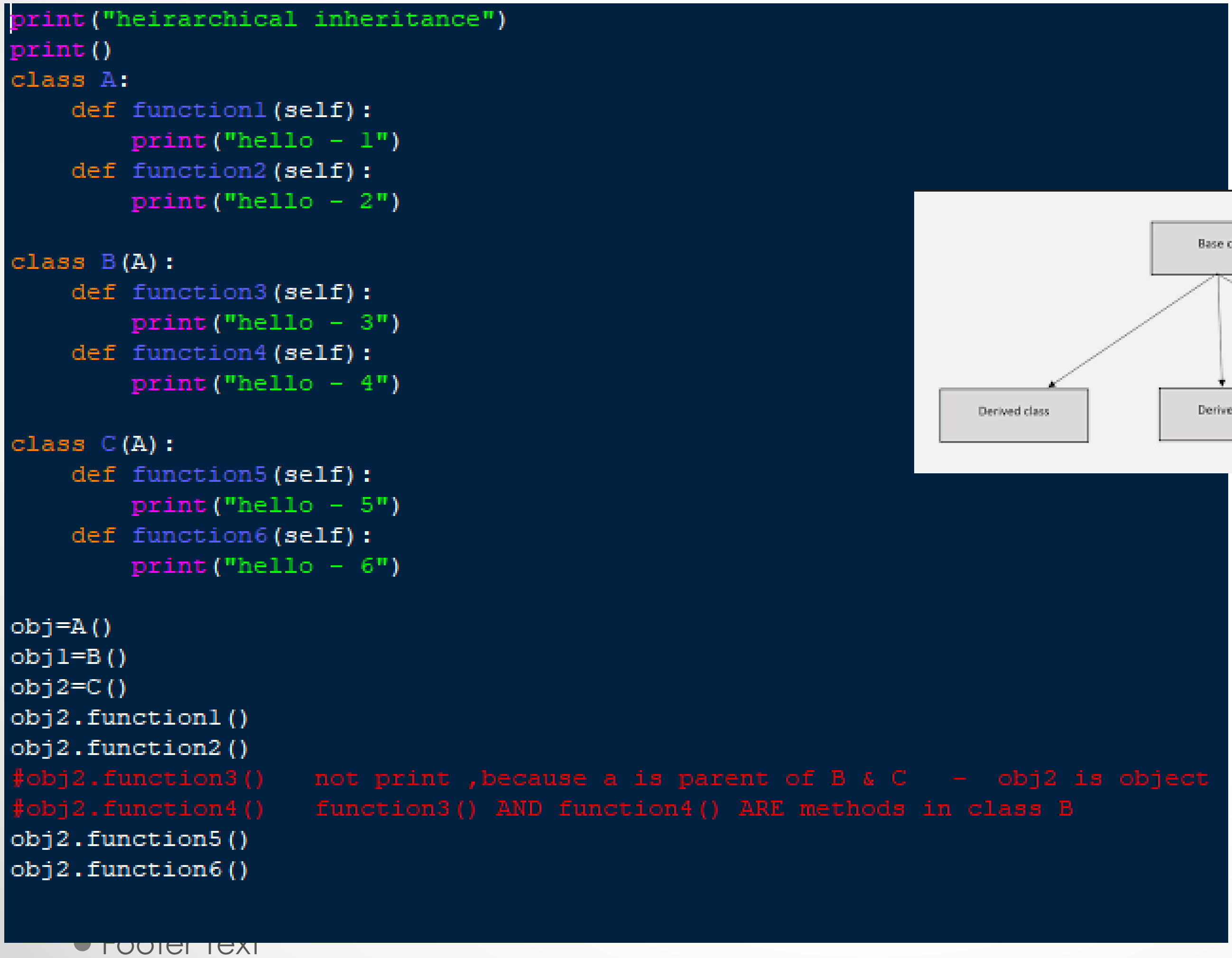


```
print("multi-level inheritance")
print()

class A:
    def function1(self):
        print("hello - 1")
    def function2(self):
        print("hello - 2")

class B(A):
    def function3(self):
        print("hello - 3")
    def function4(self):
        print("hello - 4")

class C(B):
    def function5(self):
        print("hello - 5")
    def function6(self):
        print("hello - 6")

obj=A()
obj1=B()
obj2=C()
obj2.function1()
obj2.function2()
obj2.function3()
obj2.function4()
obj2.function5()
obj2.function6()
```

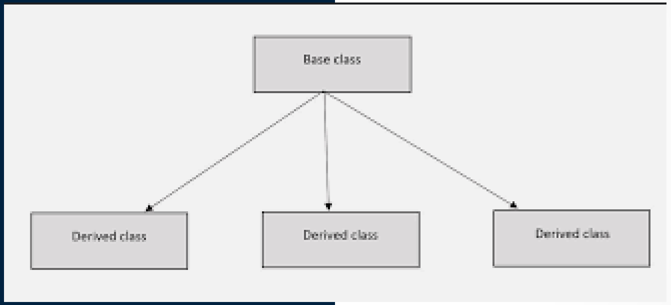
We can also inherit from a derived class. This is called multilevel inheritance.



heirarchical inheritance



```
print("heirarchical inheritance")
print()
class A:
    def function1(self):
        print("hello - 1")
    def function2(self):
        print("hello - 2")

class B(A):
    def function3(self):
        print("hello - 3")
    def function4(self):
        print("hello - 4")

class C(A):
    def function5(self):
        print("hello - 5")
    def function6(self):
        print("hello - 6")

obj=A()
obj1=B()
obj2=C()
obj2.function1()
obj2.function2()
#obj2.function3()   not print ,because a is parent of B & C   -  obj2 is object
#obj2.function4()   function3() AND function4() ARE methods in class B
obj2.function5()
obj2.function6()
```



Hybrid inheritance

Arguments and parameters in functions

Parameters:

- parameters are the variable names within the function definition.

Arguments:

- Arguments are the values that passed in the function that is called

Tkinter

- It is a GUI
- GUI-GRAPHICAL USER INTERFACE
- Python use chyth frame work chyunath -tkinter
- It is A modules - we have to import tkinter in IDLE

widget

- Label
- Entry
- Button
- Checkbox

- Scraping-
- Root/window

Place,pack,grid used to give position

- **<u>Place</u>**
- X and y axis

- font=("ariel",18,"bold","italic")
- Ariel-font type
- 18-font size

1.
2.
3. Convert Fahrenheit to Celsius
4
5.
6. Find the Root of quadratic equations
7.
8
9.
10.

Sum of a set of numbers

12. Calculation of grade based on boundary condition

13. Simple Calculator by Making Functions

14.

15.

16.

17.

18. Largest Number in a List

19. Merge Two Lists and Sort it

20. Count the number of Vowels in a string

Count the numbers of lower case letters

22.

23.