# PYTHON

→ J. M. Patel College of Commerce

→ Mrs. Soniya Sharma

# Why to Learn Python?

- Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

# Characteristics of Python

- Following are important characteristics of Python Programming −
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

# Characteristics of Python

- Following are important characteristics of Python Programming −
- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

# Features of python

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read − Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain − Python's source code is fairly easy-to-maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

# Features of python

- Databases − Python provides interfaces to all major commercial databases.
- GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows , Macintosh, and the X Window system of Unix.
- Scalable − Python provides a better structure and support for large programs than shell scripting.

# Python Installation

- Python 2.7 or higher
- For python 3.8 link - https://www.python.org/downloads/
- Download and Install.

# Python Programs

```
# This program adds two numbers


num1 = 1.5

num2 = 6.3


# Add two numbers

sum = num1 + num2


# Display the sum

print(“addition of two numbers”,sum)
```

# Python Programs

```
# Python Program to find the area of triangle

a = 5

b = 6

c = 7

# Uncomment below to take inputs from the user

# a = float(input('Enter first side: '))

# b = float(input('Enter second side: '))

# c = float(input('Enter third side: '))

# calculate the semi-perimeter

s = (a + b + c) / 2

# calculate the area

area = (s*(s-a)*(s-b)*(s-c)) ** 0.5

print('The area of the triangle is %0.2f' %area)
```

# Python Programs

```
#find area of circle

r=int(input("enter a number"))

print("area of circle is",3.14*r*r)
```

# Python Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Comments starts with a #, and Python will ignore them:

Example

```
#This is a comment

print("Hello, World!")
```

You can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""

This is a comment

written in

more than just one line
```

# Python Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:

Arithmetic operators

Assignment operators

Comparison operators

Logical operators

Identity operators

Membership operators

https://www.w3schools.com/python/python_operators.asp

# Python Conditions and If statements

Example

**If statement:**

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

# Python Conditions and If statements

```
a = 33

b = 200

if b > a:

print("b is greater than a") # you will get an error if indentation is wrong
```

**********

**Elif**

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33

b = 33

if b > a:

  print("b is greater than a")

elif a == b:

  print("a and b are equal")
```

# Python Conditions and If statements

**Else**

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

# Python Conditions and If statements

**And**

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

# Python Conditions and If statements

**Nested If**

You can have if statements inside if statements, this is called nested if statements.

Example

```
#find greatest number between 3 numbers
n1=int(input("enter a number 1"))
n2=int(input("enter a number 2"))
n3=int(input("enter a number 3"))
if(n1>n2):
    if(n1>n3):
        print("greatest number is: ",n1)
elif(n2>n1):
    if(n2>n3):
        print("greatest number is:",n2)
if(n3>n1 and n3>n2):
    print("greatest number is n3",n3)
```

# Python Collections

Following are the collection data types in the Python programming language:

- List
- Tuple
- Dictionary
- Array

**List**

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

**Example**

Create a List:

```
thislist = ["", "banana", "cherry"]
print(thislistapple)
```

# Python Collections

**Access Items**

You access the list items by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])

print(thislist[-1])
```

**Example**

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print(thislist[:4])

print(thislist[2:5])

print(thislist[2:])

print(thislist[-4:-1])
```

# Python Collections

**Change Item Value**

```
fruit = ["apple", "banana", "cherry"]
fruit[1] = "blackcurrant"
print(fruit)
```

**Loop Through a List**

```
fruit = ["apple", "banana", "cherry"]
for x in fruit:
  print(x)
```

**Check if Item Exists**

```
fruit = ["apple", "banana", "cherry"]
if "apple" in fruit:
  print("Yes, 'apple' is in the fruits list")

print(len(fruit)) #finds length of list
```

**Sorting list**

```
thislist = list(("peru","apple", "banana", "cherry"))

thislist.sort()

print(thislist)
```

# Python Collections

**Add Items**

```
fruit = ["apple", "banana", "cherry"]

fruit.append("orange") #inserting at end
print(fruit)

fruit.insert(1, "orange") #inserting at particular location
print(fruit)
```

**Remove Item**

```
fruit = ["apple", "banana", "cherry"]
fruit.remove("banana")
print(fruit)

fruit = ["apple", "banana", "cherry"]
fruit.pop() #removes the specified index or last item if index is not specified
print(fruit)

fruit = ["apple", "banana", "cherry"]
del fruit[0] # removes the specified index
print(fruit)

del fruit # can also delete the list completely
```

# Python Collections

```
fruit = ["apple", "banana", "cherry"]
fruit.clear()
print(fruit) #empties the list
```

```
mylist = thislist.copy() #copies a list
print(mylist)
```

**Join two list**

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

**Append list2 in list1**

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

# Python Tuple

- A tuple is a collection of objects which ordered and immutable.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed and tuples use parentheses, whereas lists use square brackets.

```
# tuple demonstration

fruit=("apple","banana","orange","papaya","kiwi","mango","melon")

print("Tuple : ",fruit)

print("2nd : ",fruit[2])

print("Last : ",fruit[-1])

print("2:5 range ",fruit[2:5])

print("-4 to -1 range ",fruit[-4:-1])
```

# Python Tuple

```
# tuple are immutable

fruit=("apple","banana","orange","papaya","kiwi","mango","melon")

# fruit[1]="watermelon" --Error

listfruit=list(fruit)

listfruit[1]="watermelon"

print(listfruit)


#loop through tuple

fruit=("apple","banana","orange","papaya","kiwi","mango","melon")

for i in fruit:

    print(i)
```

# Python Tuple

```
#check item existance (in / not in)
fruit=("apple","banana","orange","papaya","kiwi","mango","melon")
item=input("enter fruit name: ")
if item in fruit:
    print("yes it is there in fruit")
else:
    print("No it is not there in fruit")


# len() and type()
fruit=("apple","banana","orange","papaya","kiwi","mango","melon")
print("length : ",len(fruit))
print("type : ",type(fruit))
```

# Python Tuple

**#concat tuples in another tuple**

```
fruit=("apple","banana","orange","papaya","kiwi","mango","melon")
flower=("rose","tulip","sunflower")
ff=fruit+flower
print(ff)
```

**# count()**

```
flower=("rose","tulip","rose","sunflower","rose")
n=flower.count("rose")
print("rose came :",n," times")
```

# Python Dictionary

**A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.**

```
mydictn= {"car":"duster",
        "model":"top model",
        "year":2019}
print(mydictn) # will print the entire dictionary
print(mydictn["car"]) #will print the value
print(mydictn.get("car"))
mydictn["car"]="ciaz" #change the value
```

**Looping through dictionary**

```
print("printing keys")
for x in mydictn:
    print(x) #only keys
print("printing values")
for x in mydictn:
    print(mydictn[x]) # only values
for x in mydictn.values():
    print(x) # only values
```

# Python Dictionary

```
#displaying contents of dictionary
mydictn= {"car":"duster",
          "model":"top model",
          "year":2019}
print(mydictn)
for x, y in mydictn.items():
    print(x,y)
    if(x=="year" and y==2019):
        print("You get insurance of 5 lac")
 print(len(mydictn)) #printing length of dictionary
#adding key value to the dictionary
mydictn["color"]="brown"
print(mydictn)
print(len(mydictn))
```

# Python Dictionary

```
mydictn.pop("year") #remove element from dictionary

mydictn.popitem() # removes last inserted value

del mydictn["price"] #delete key value pair

del mydictn #delete entire dictionary

mydictn.clear() # empties dictionary

mydic2=mydictn.copy() # copying contents

mydic2=dict(mydictn) #copying contents
```

**#Nested dictionary**

```
car = { "name" : "duster", "year": 2018 }

bike ={ "name":"bullet","year" : 2000}

cycle={"name":"honda", "year":2020}

vehicals =  {"child1":car,"child2":bike,"child3":cycle}

print(vehicals)
```

**Dictionary key property**

1. key cannot be duplicated.



2. value does not exist without key and vice versa

# Array

**Arrays are used to store multiple values in one single variable**:

cars = ["Ford", "Volvo", "BMW"]

**An array is a special variable, which can hold more than one value at a time.**

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

**The solution is an array!**

An array can hold many values under a single name, and you can access the values by referring to an index number.

# Array

**Access the Elements of an Array**

You refer to an array element by referring to the *index number*.

x = cars[0]

cars[0] = "Toyota“ # **modifies** value

x = len(cars) # gives **length** of array

#**looping** array element

```
for x in cars:
  print(x)
```

cars.append("Honda") # **adds** array element

cars.pop(1) # **remove** array element

cars.remove("Volvo")

# Python Loops

**Python has two primitive loop commands:**

- while loops
- for loops

**Example**

Print i as long as i is less than 6:

```
i = 1
while i < 6:
        print(i)
        i += 1
```

**Example (break statement)**

Exit the loop when i is 3:

```
i = 1
while i < 6:
        print(i)
        if i == 3:
            break
        i += 1
```

# Python Loops

**Example (continue statement)**

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
        i += 1
        if i == 3:
            continue
        print(i)
```

# Program to practice

1. **Display sum of first 10 numbers. (H.W)**
2. **Find sum of even numbers in first 20 numbers. (H.W)**
3. **Find factorial of a number. (H.W)**
4. **Find a number is prime or not (Sloved)**
5. **Find a number is even or odd. (H.W)**
6. **Input 5 subjects marks, find percentage and give proper grade. (H.W)**
7. **Find reverse of a number. (H.W)**
8. **Swap two numbers. (Sloved)**
9. **Generate Fibonacci series. (Sloved)**
10. **Check number is armstrong or not. (Sloved)**
11. **Check number is palindrome or not. (Sloved)**
12. **Enter 10 numbers and display even numbers. (Sloved)**

# Program to practice

**# 4. Find a number is prime or not (Sloved)**

```
n=int(input("enter a number: "))
i=2
flag=1
while (i<n):
      if(n%i==0):
           flag=0
       break
   i=i+1
if(flag==1):
   print("number is prime")
else:
   print("number is not prime")
```

# Programs to practice

## # 9. Fibonacci series

```
#Fibonacci series is like 0 1  1  2  3   5  8  13…

n=int(input("how many terms of fibonacci
series u want :"))

a=0

b=1

if(n==1):

   print("  ",a)

if(n==2):

   print("  ",a)

   print("  ",b)

elif(n>=3):

   print("  ",a)

   print("  ",b)

   i=3

   while(i<=n):

      c = a + b

         print("  ",c)

         a = b

         b = c

         i= i+1

print("finished")
```

# Program to practice

```
# 10. Check number is armstrong or not.

n=int(input("enter a number"))

ar=n

sum=0

while (n>0):

    a=n%10

    sum=a*a*a+sum

    n=int(n/10)

if(sum==ar):

    print("number is armstrong")

else:

    print("number is not armstrong")
```

# Program to practice

**# 11. Check the number is palindrome or not.**

```
n=int(input("enter a number"))
pal=n
rev=0
while(n>0):
    a=n%10
    rev=rev*10+a
    n=int(n/10)
print("the reverse number is: ",rev)
if(pal==rev):
    print("the number is palindrom")
else:
    print("number is not palindrom")
```

# Program to practice

**# 12. enter 10 numbers and display even numbers**

```
i=1
num=[]
while(i<=5):
    num.insert(i,(int(input("enter a number : "))))
    i=i+1
print("the list is")
for i in num:
    if(i%2==0):
        print(i)
```

# Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

```
#function demonstration
def addition(c,d):
    return c+d
a=int(input("enter 1 number"))
b=int(input("enter 2nd number"))
x=addition(a,b)
print("addition of two numbers is : ",x)
```

# Python Functions

```
#function demonstration
def dis_name(fname):
    print(fname+" yadav")
dis_name("brijesh")
dis_name("Neeraj")
dis_name("ajay")
dis_name("aman")
dis_name("vikas")
```

A **parameter** is the variable listed inside the parentheses in the function definition.

An **argument** is the value that is sent to the function when it is called.

# Python Functions

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a *tuple* of arguments, and can access the items accordingly:

```
#Arbitrary Arguments passing

def dis_name(*fname):
    print(fname[0]+" yadav")
    print(fname[1]+" yadav")
    print(fname[2]+" yadav")
    print(fname[3]+" yadav")
    print(fname[4]+" yadav")


dis_name("brijesh","Neeraj","ajay","aman","vikas")
```

# Python Functions

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

```
#default argument
def area(r=5):
    print("radius : ",r)
    print("area of circle is :",3.14*r*r)
area(12)
area(3)
area(6)
area()
area()
```

# Python Functions

```
#passing list as argument
def my_fun(food):
    for x in food:
        print(x)
fruit=["apple","kiwi","cherry"]
my_fun(fruit)


#passing dictionary as argument
def my_fun(food):
    for x in food:
        print(x," : ",food[x])


fruit={1:"apple",2:"kiwi",3:"cherry"}
my_fun(fruit)
```

# Python Class & object

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- **The __init__() Function**
- The examples above are classes and objects in their simplest form, and are not really useful in real life applications.
- To understand the meaning of classes we have to understand the built-in __init__() function.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:
- class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  p1 = Person("John", 36)

  print(p1.name)
  print(p1.age)

# Object Methods

```
# class and object

class person:

    def __init__(self,name,age):

        self.name = name

        self.age = age

    def show(self):

        print(" hi ",self.name)

        print("your age is : ",self.age)

p1=person("Manish",45)

p2=person("priya",23)

p1.show()

p2.show()
```

# Object Methods

The self Parameter

The `self` parameter is a reference to the current instance of the class, 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 function in the class:

```
# class and object
class person:
   age=90

p1=person()
print("age is ",p1.age)

p1.age=45

print("age is ",p1.age)

del p1.age

print("age is ",p1.age)
```

# Programs to practice (H.W)

1. Find area of circle using class and object.

2. Input student name and 3 subject marks calculate their percentage and give grade using class and object.

3. Input employee and their salary details such as basic, hra, ta , da if total salary is greater than 60000 then ask them to pay the tax (using class object)
4. Find reverse number by using class and object.

5. Find area of rectangle and square using different functions of class.
6. Create class student and **employee** (child of student), input name, marks details and salary details by using employee class object only. (Through inheritance).
7. Find square, cube of number using inheritance.

# Inheritance

**Python Inheritance**

Inheritance allows us to define a class that inherits all the methods and properties from another class.

**Parent class** is the class being inherited from, also called base class.

**Child class** is the class that inherits from another class, also called derived class.

```
#inheritance

class student:

    def __init__(self,fname,lname):

        self.fname=fname

        self.lname=lname

    def show(self):

        print("Hi ",self.fname," ",self.lname)

class syit(student):

    pass

s1=syit("vikas","yadav")

s1.show()
```

# Inheritance

```
#inheritance example 2
class student:
    def __init__(self,fname,lname):
        self.fname=fname
        self.lname=lname
    def show(self):
        print("Hi ",self.fname," ",self.lname)
class syit(student):
    def __init__(self,fn,ln,ayear):
        self.ayear=ayear
        super().__init__(fn,ln)
    def display(self):
        print("academic year is :",self.ayear)
s1=syit("priya","gupta",2020)
s1.show()
s1.display()
```

# Inheritance

```
#inheritance example Multilevel Inheritance
class A:
    def showA(self):
        print("hello i am class A")
class B(A):
    def showB(self):
        print("hello i am class B")
class C(B):
    def show(self):
        print("hello i am class C")


obj=C()
obj.showA()
obj.showB()
obj.show()
```

# Inheritance

```
#inheritance example Multilevel Inheritance

class Employee:
    def getEmp(self,eid,ename):
        self.eid=eid
        self.ename=ename
    def showEmp(self):
        print("employee id : ",self.eid)
        print("employee name : ",self.ename)

class Product(Employee):
    def product_details(self,pid,pname):
        self.pid=pid
        self.pname=pname
        print("product id :",self.pid)
        print("Product name :",self.pname)

class customer(Product):
    def cust_details(self,cid,cust_name):
        self.cid=cid
        self.cust_name=cust_name
        print("customer id: ",self.cid)
        print("customer name: ",self.cust_name)
c1=customer()
c1.cust_details(111,"hari")
c1.getEmp(999,"mohan")
c1.showEmp()
c1.product_details(1001,"maggi")
print("program over")
```

# Iterators

- An iterator is an object that contains a countable number of values.
- An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
- `__iter__()` and `__next__()`
- Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which you can get an iterator from.
- All these objects have a `iter()` method which is used to get an iterator.
- E.g.

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

- Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"
myit = iter(mystr)
for x in myit:
        print(x)
```

# Strings

- String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello".

- Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

- Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

- Slicing

You can return a range of characters by using the slice syntax.



Specify the start index and the end index, separated by a colon, to return a part of the string.

# Strings

- b = "Hello, World!"
  print(b[2:5])

Negative Indexing

- Use negative indexes to start the slice from the end of the string:

b = "Hello, World!"
print(b[-5:-2])

- String Length

To get the length of a string, use the `len()` function.

a = "Hello, World!"
print(len(a))

- The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!“
```

print(a.upper()) #display string in upper case
Print(a.lower()) #display string in lower case
print(a.replace("l", "J")) #replaces l with j

# Strings

- The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

- **Check String**
- Eg.

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
```

- **String Concatenation**

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
```

- **Strings and numbers can be combined by using format() function.**
- Eg.

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))

# output – My name is John, and I am 36
```

# Strings

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

- **Escape Character**

To insert characters that are illegal in a string, use an escape character.
An escape character is a backslash \ followed by the character you want to insert.
An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)

# output – We are the so-called “Vikings” from the north.
```

# Scope

A variable is only available from inside the region it is created. This is called **scope**.

**Local Scope**

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

A variable created inside a function is available inside that function:

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

**Global Scope**

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

# Scope

**Global Scope**

A variable created outside of a function is global and can be used by anyone:

```
x = 300

def myfunc():
  print(x)
myfunc()
print(x)
```

**If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):**

The function will print the local x, and then the code will print the global x:

```
x = 300
def myfunc():
  x = 200
  print(x)
myfunc()
print(x)
```

# Math Functions

**Built-in Math Functions:**

- The `min()` and `max()` functions can be used to find the lowest or highest value in an iterable:

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

- The `abs()` function returns the absolute (positive) value of the specified number:

```
x = abs(-7.25)

print(x)
```

- The pow(x,y) returns value of x to the power of y.

```
x = pow(4, 3)

print(x)
```

# Math Module

- Python has also a built-in module called `math`, which extends the list of mathematical functions.

To use it, you must import the `math` module:

```
import math
x=math.sqrt(100)
print("square root is :",x)
```

- The `math.ceil()` method rounds a number upwards to its nearest integer, and the `math.floor()` method rounds a number downwards to its nearest integer, and returns the result:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

- Pi value in math module
- Print(math.pi)

# File Handling

- File handling is an important part of any web application.
- Python has several functions for creating, reading, updating, and deleting files.
- The key function for working with files in Python is the `open()` function.
- The `open()` function takes two parameters; ***filename***, and *mode*.
- There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

- **Reading file (file is saved at same location as program)**

```
f=open("myfile.txt","r")
print(f.read())
```

- **Reading file (If the file is located in a different location)**

```
f=open("D://Soniya/abc.txt","r")
print(f.read())
```

- **To read one line of file**

```
f=open("D://Soniya/abc.txt","r")
print(f.readline())
print(f.readline())
```

# File Handling

- **Looping through the file line by line**

```
f=open("D://Soniya/abc.txt","r")
for x in f:
    print(x)
```

- **Close Files**

It is a good practice to always close the file when you are done with it for security and safety.

```
f.close()
```

- **Write to an Existing File**

To write to an existing file, you must add a parameter to the `open()` function:
`"a"` - Append - will append to the end of the file
`"w"` - Write - will overwrite any existing content

- **Append contents to the file**

```
f=open("D://Soniya/abc.txt","a")
f.write("i like to go to party and have fun")
f.write("i also belive in hard work")
f.close()
```

# File Handling

- **Writing contents to the file**

```
f=open("D://Soniya/abc.txt","w")
f.write("i like to go to party and have fun")
f.write("\n i also belive in hard work")
f.close()
```

- **Append (a) VS Write (w)**
- Append mode (a) add new contents to the previous data without deleting it.
- Write mode (w) add new contents to the file by deleting previous contents.

- **Delete a File**

To delete a file, you must import the OS module:

```
import os
os.remove("D://Soniya/abc.txt")
print("file deleted plz check from location")
```

- **Deleting directory /folder:**

```
import os
os.rmdir("D://soniya/demo_syit")
print("directory deleted plz check from location")
```

**We can delete only empty directories.**

# Exceptions

- The `try` block lets you test a block of code for errors.
- The `except` block lets you handle the error.
- The `finally` block lets you execute code, regardless of the result of the try- and except blocks.
- The raise block lets you raise / throw / generate exception.

```
#The try block will generate an error, because x is not defined:
try:
        print(x)
        print("hello:")
except:
        print("exception occurred")
```

(Since the try block raises an error, the except block will be executed.)

- #The try block will generate a NameError, because x is not defined:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

# Exceptions

- **ELSE**

#The try block does not raise any errors, so the else block is executed:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong, when no exception arrised")
```

- **Finally : The finally block gets executed no matter if the try block raises any errors or not.**

```
try:
  print("hello")
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

- This can be useful to close objects and clean up resources

# Exceptions

- **Raise exception**

```
x = -10

if x > 0:
        print(x)
else:
        raise Exception("Sorry, no numbers below zero“)
```

- **TypeError**

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
else:
        print("no problem")
```

# Exceptions

- **Raise exception**

```
x = -10

if x > 0:
        print(x)
else:
        raise Exception("Sorry, no numbers below zero“)
```

- **TypeError**

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
else:
        print("no problem")
```

# Exceptions

- **Practice program**

```
a=int(input("enter a number"))
b=int(input("enter second number"))

if b==0:
    raise Exception("division by zero is not possible")
else:
    c=a/b
    print("Division of two numbers is",c)
```

# Modules

- A module is the same as a code library.
- A file containing a set of functions you want to include in your application.

**Step 1 : To create a module**

- Save the below program with name - mygreet.py . It is the module which contains two functions.

```
def greet(name):
    print("Hello, "+name)
    print("how are you")

def revs(n):
    print("you passed me: ",n)
    rev=0
    while n>0:
        a=n%10
        rev=rev*10+a
        n=int(n/10)
    print("the reverse number is :", rev)
```

**Step 2: Importing a module**

```
import mygreet

mygreet.greet("Nidhi")
mygreet.revs(345)
```

# Modules

## Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

```
import mygreet as mg

mg.greet("Nidhi")
mg.revs(345)
```

## Python Date, Time, Calendar Module

### #program 1

```
import datetime
x = datetime.datetime.now()
print(x)
```

### #program 2

```
import time;
localtime = time.localtime(time.time())
print ("Local current time :", localtime)
```

# Modules

**#program 3**

```
import calendar

cal = calendar.month(2020, 9)
print("Here is the calendar:")
print(cal)
```

## Random module

**random.random():** Generates a random float number between 0.0 to 1.0. The function doesn't need any arguments.

**random.randint():** Returns a random integer between the specified integers.

```
import random
print(random.randint(1,100))
print(random.random())
```