Meaning of Testing

In [ ]:
```
Meaning of Testing --> Testing is a process by which we can validate,verify and check our applications/softwares.
                   --> it is basically used to check how much your software is upto mark.
```

Types of Testing

In [ ]:
```
Types of Testing:

    Unit Testing         --> It focuses on the smallest unit of software design. In this, we test an individual unit or group of
                             interrelated units.
    Integration Testing --> The objective is to take unit-tested components and build a program structure that has been
                             dictated by design.
                        --> Integration testing is testing in which a group of components is combined to produce output.
    System Testing       --> System testing is testing conducted on a complete integrated system to evaluate the system's
                             compliance with its specified requirements.
    Smoke Testing        --> Smoke Testing is a software testing method that determines whether the employed build is
                             stable or not.
    Acceptance Testing  --> Acceptance Testing is the last phase of software testing performed after System Testing and
                             before making the system available for actual use.
```

assert keyword

In [ ]:
```
assert keyword --> Assert is a keyword that is used for debugging the code.
               --> Assert will check the condition if condition is true then assert will return True else assert will
                   return an Assertion Error
               --> You can write a message with assertion error as well if the assert will return False
```

Example

In [5]:
```
x="Aman"
assert x=="Aman1","x is not Aman"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 x="Aman"
----> 2 assert x=="Aman1","x is not Aman"

AssertionError: x is not Aman
```

In [1]:
```
x="Aman"
assert x=="Aman"
```

Framework Present in Python for Testing

In [ ]:
```
Frameworks that are present in Python:
    1.Unittest
    2.pytest
    3.Doctest
    4.Testify
    ...
    ...
    ...
    ...
```

About Pytest

In [ ]:
```
--> pytest is framework in python that is used to write our own test cases based on certain inputs or criteria using python
    programming language.
```

Why pytest

In [ ]:
```
Why pytest :
    --> very easy to start beacuse it has very simple syntax
    --> Skip Test
    --> Open source
    --> Automatically detetct tests
```

Important Note

In [ ]:
```
--> You cannot directly use pytest for using pytest you need to use either teriminal or command prompt
--> Command --> pytest file_name.py
```

Example -1

In [ ]:
```
def add(x,y):
    return x+y
def product(x,y):
    return x*y

def test_add():
    assert add(7,3)==10
    assert add(9)==10
    assert add(5)==7

def test_product():
    assert product(5,5)==25
    assert product(5)==25
    assert product(7)==35
```

Example -2

In [10]:
```
def factorial(n):
    if n<0:
        return "Negative Number"
    elif n==0:
        return 1
    elif n==1:
        return 1
    else:
        fact=1
        for i in range(2,n+1):
            fact=fact*i
        return fact
def test_factorial():
    assert factorial(0)==1
    assert factorial(1)==1
    assert factorial(5)==120
    assert factorial(-9) == "Negative Number"
```

pdb (Python Debugger)

In [ ]:
```
pdb --> It is also a module of python that will help you to debug your.pbm is internally makes (basic debugger functions)
        and cmd. pdb is stand for Python debugger.
```

Example

In [ ]:
```
import pdb
def add(x,y):
    return x+y
x=int(input("Enter a"))
y=int(input("Enter b"))
z=add(x,y)
print(z)
```

On Command Prompt:

In [ ]:
```
(base) C:\Users\praty>python pdbdemo.py
Enter a10
Enter b10
20

(base) C:\Users\praty>python pdbdemo.py
Enter a10
Enter b10
20

(base) C:\Users\praty>python -m pdb pdbdemo.py
> c:\users\praty\pdbdemo.py(2)<module>()
-> """
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    c              d          h         list      q        rv       undisplay
a      cl             debug      help      ll        quit     s        unt
alias  clear          disable    ignore    longlist  r        source   until
args   commands       display    interact  n         restart  step     up
b      condition      down       j         next      return   tbreak   w
break  cont           enable     jump      p         retval   u        whatis
bt     continue       exit       l         pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb

(Pdb) help next
n(ext)
        Continue execution until the next line in the current function
        is reached or it returns.
(Pdb) help continue
c(ont(inue))
        Continue execution, only stop when a breakpoint is encountered.

(Pdb) n
> c:\users\praty\pdbdemo.py(7)<module>()
-> import pdb
(Pdb) n
> c:\users\praty\pdbdemo.py(8)<module>()
-> def add(x,y):
(Pdb) n
> c:\users\praty\pdbdemo.py(10)<module>()
-> x=int(input("Enter a"))
(Pdb) n
Enter a10
> c:\users\praty\pdbdemo.py(11)<module>()
-> y=int(input("Enter b"))
(Pdb) n
Enter b10
> c:\users\praty\pdbdemo.py(12)<module>()
-> z=add(x,y)
(Pdb) n
> c:\users\praty\pdbdemo.py(13)<module>()
-> print(z)
(Pdb) n
20
--Return--
> c:\users\praty\pdbdemo.py(13)<module>()->None
-> print(z)
(Pdb) n
--Return--
> <string>(1)<module>()->None
(Pdb) n
The program finished and will be restarted
> c:\users\praty\pdbdemo.py(2)<module>()
-> """
(Pdb)
```

pdb.set_trace()

In [ ]:
```
--> pdb.set_trace() begin the debugger at this line when the file is run normally
```

Example

In [11]:
```
import pdb
def add(x,y):
    return x+y
x=int(input("Enter a")) #10
pdb.set_trace()
y=int(input("Enter b"))
z=add(x,y)
print(z)
```

```
--Return--
None
> c:\users\praty\appdata\local\temp\ipykernel_15700\2573240595.py(5)<cell line: 5>()

--KeyboardInterrupt--


KeyboardInterrupt: Interrupted by user
30
```

Generators

In [ ]:
```
Generators --> Generators are the functions that are used to generate a sequence of values.
           --> Generators are more faster as compare to other data type.
           --> yield is a keyword to generate a sequence of values
           --> next is used to get the value of generated object.
```

Example

In [14]:
```
def generator():
    yield 1
    yield 2
    yield 3
x = generator()
print(next(x))
print(next(x))
print(next(x))
```

```
1
2
3
```

Advantages of Generators:

In [ ]:
```
#Advantages of Generators.
1.When compared to other iterators generators are easy to use
2.Improvoes memeory utilization and performance
3.Generator object is best suitable forr reading the large amount of data
```

Generator vs Normal Collections

In [8]:
```
y=(x*x for x in range(10000000000000000000000000))
```

In [ ]:
```
y=[x*x for x in range(10000000000000000000000000)]
```