Meaning of Testing - process of evauialting, verfiying the softwares/codes. It is basically Used to check how much our application is upto mark. Types of Testing: Manual Testing Automation Testing ********** Types of Testing: ********** Unit tests. --> Integration tests. ... Functional tests. ... End-to-end tests. ... Acceptance testing. .. . Performance testing. ... Smoke testing.

In [ ]:
```
#What is assert keyword?
Assert keyword is used for debugging the code.
Assert keyword lets you test if a condition in your code returns true if not then the program
will return assertion error
You can write a message to be written while the assert is false by giving comma
```

In [5]:
```
x="Aniket"
assert x=="Aniket1",123
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 x="Aniket"
----> 2 assert x=="Aniket1",123

AssertionError: 123
```

In [ ]:
```
#What are the Testing Frameworks we have in python?
1.Unittest
2.Pytest
3.DocTest
4.Testify
```

In [ ]:
```
#What is Pytest?
Pytest is a testing framework of Python.that allows user to write your own test cases using python
programming.
```

In [ ]:
```
#How to use pytest in our code
1.very easy to start beacuse it has very simple syntax
2.Skip Test
3.Open source
4.Automatically detetct tests
```

In [6]:
```
#Note: You cannot directly use pytest for that you need to use either teriminal or command prompt
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
#Command --> pytest file_name.py
```

In [ ]:
```
def fact(x):
    if x==0:
        return 1
    elif x==1:
        return 1
    else:
        fact=1
        for i in range(1,x+1):
            fact=fact*i
        return fact



def test_fact():
    assert fact(0)==0
    assert fact(5)==120
    assert fact(6)==720
#Command --> pytest file_name.py
```

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.
we can only use pdb only in command prompt or terminal.
```

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)
```

In [ ]:
```
"""
Created on Fri Sep 16 20:38:00 2022

@author: praty
"""
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)
```

In [ ]:
```
set_trace is used to give the breakpoint to the code. basically it is a point from which
debugging is started automatically.
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)
```

Generators

In [ ]:
```
Generator --> is used to genrate a sequence of values
We can write genertor function like ordinary function but it is using yield keyword
to return values
```

In [12]:
```
def gen():
    yield "A"
    yield "B"
    yield "C"
g=gen()
print(type(g))
print(next(g))
print(next(g))
print(next(g))
```

```
<class 'generator'>
A
B
C
```

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

In [ ]:
```
#Generator  vs Normal Collections with respect to Memory Utilization
#Nomral Collection
y=[x*x for x in range(1000000000000)]
print(y[0])
```

In [ ]:
```
y=[x*x for x in range(10000000000000000000)]
print(y[0])
```

In [ ]:
```
y=(x*x for x in range(10000000000000000000))
print(y[0])
```

In [ ]:
```

```