



# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing and Debugging

---

**Prof. Sindhu R Pai**

PCPS Theory Anchor - 2024

Department of Computer Science and Engineering

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing – Pytest

---



### Pytest

- Pytest is a robust testing framework for Python.
- It allows users to write test codes using Python programming language.
- It helps to write tests from simple unit tests to complex functional tests.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing - Pytest

---



### Advantages of Pytest

- Free and open source
- Simple syntax - very easy to start with
- Run multiple tests in parallel, which reduces the execution time of the test suite
- Automatically detect test file and test functions, if not mentioned explicitly
- Allows to skip a subset of the tests during execution
- Allows to run a subset of the entire test suite

### Features of Pytest

1. Does not require API to use
2. Provides useful plugins
3. Can be written as a function or method
4. Gives useful failure information without the use of debuggers
5. Can be used to run doc tests and unit tests

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing - Pytest

---



### Pytest – Environmental Setup

1. Open command prompt
2. Change directory to the location where Python is installed
3. Type command

```
pip install pytest
```

4. Confirm the installation

```
pytest -h
```

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing - Pytest




### Pytest – Environmental Setup

```
C:\Users\SOWMYA SHREE P>cd C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311

C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311>pip install pytest
Collecting pytest
  Downloading pytest-7.4.3-py3-none-any.whl.metadata (7.9 kB)
Collecting iniconfig (from pytest)
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting packaging (from pytest)
  Downloading packaging-23.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pluggy<2.0,>=0.12 (from pytest)
  Downloading pluggy-1.3.0-py3-none-any.whl.metadata (4.3 kB)
Collecting colorama (from pytest)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading pytest-7.4.3-py3-none-any.whl (325 kB)
   _____ 325.1/325.1 kB 3.3 MB/s eta 0:00:00
Downloading pluggy-1.3.0-py3-none-any.whl (18 kB)
Downloading packaging-23.2-py3-none-any.whl (53 kB)
   _____ 53.0/53.0 kB 2.7 MB/s eta 0:00:00
Installing collected packages: pluggy, packaging, iniconfig, colorama, pytest
Successfully installed colorama-0.4.6 iniconfig-2.0.0 packaging-23.2 pluggy-1.3.0 pytest-7.4.3

C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311>pytest -h
```

### Pytest – Example

1. Create a new directory (say “Automation”) and navigate into the directory in the command line.
2. Create a file `pytestExample.py` 
3. Run the file using the command  
`pytest pytestExample.py`

```
import math

def testsqrt():
    num = 25
    assert math.sqrt(num) == 5

def testsquare():
    num = 7
    assert 7*7 == 40

def testequality():
    assert 10 == 11
```

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Testing - Pytest



### Pytest – Example (Output)

```
C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311\Automation>pytest pytestExample.py
===== test session starts =====
platform win32 -- Python 3.11.5, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311\Automation
collected 3 items

pytestExample.py .FF

===== FAILURES =====
----- testsquare -----

    def testsquare():
        num = 7
>       assert 7*7 == 40
E       assert (7 * 7) == 40

pytestExample.py:9: AssertionError
----- testequality -----

    def testequality():
>       assert 10 == 11
E       assert 10 == 11

pytestExample.py:12: AssertionError
===== short test summary info =====
FAILED pytestExample.py::testsquare - assert (7 * 7) == 40
FAILED pytestExample.py::testequality - assert 10 == 11
===== 2 failed, 1 passed in 0.06s =====

C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311\Automation>
```



# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest

---



### Doctest

- It is a module included in the Python programming language's standard library.
- It allows the easy generation of tests based on output from the standard Python interpreter shell.
- It **finds patterns in the *docstring*.**
- *Docstrings* - provides description of a class or a function to provide a better understanding of the code. Also, used for testing purposes using doctest module.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest

---



### Need for Doctest

- To check that a module's docstrings are up-to-date (to ensure code still work as documented).
- To perform regression testing (verifying the changes made to the code will not impact the existing functionalities of the Software).

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest



### Steps to write a function with doctest

1. Import the `doctest` module.
2. Write the function with docstring.
3. Inside the docstring, write the following two lines for testing the function.

```
>>>function_name(args)
```

Expected Output

4. Write the function logic(Coding).
5. Call the `doctest.testmod(name= function_name, verbose=True)`

If 'verbose' is set to False(default), output will be shown in case of failure only, not in the case of success.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest



### Example1: Illustrating the testcase-pass

```
# import testmod for testing a function
```

```
from doctest import testmod
```

```
# define a function to test
```

```
def fact(n):
```

```
    '''
```

```
    >>> fact(5)
```

```
    120
```

```
    >>> fact(0)
```

```
    1
```

```
    '''
```

```
    if n==0:
```

```
        res=1
```

Lines for testing the function.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest



### Example1 (Contd...)

```
else:  
    res=n*fact(n-1)  
return res
```

# call the testmod function

```
testmod(name='fact', verbose = True)
```

### Output:

```
Trying:  
    fact(5)  
Expecting:  
    120  
ok  
Trying:  
    fact(0)  
Expecting:  
    1  
ok  
1 items had no tests:  
    fact  
1 items passed all tests:  
    2 tests in fact.fact  
2 tests in 2 items.  
2 passed and 0 failed.  
Test passed.
```

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest



### Example2: Illustrating the testcase-failed

```
# import testmod for testing a function
```

```
from doctest import testmod
```

```
# define a function to test
```

```
def fact(n):
```

```
    """
```

```
    >>> fact(5)
```

```
    120
```

```
    >>> fact(0)
```

```
    1
```

```
    """
```

```
    if n==0:
```

```
        res=1
```

Lines for testing the function.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Function testing with Doctest



### Example2 (Contd...)

```
else:
    res=fact(n-1) #Wrong logic to compute factorial
    return res
```

# call the testmod function

```
testmod(name='fact', verbose = True)
```

### Output:

```
Failed example:
    fact(5)
Expected:
    120
Got:
    1
Trying:
    fact(0)
Expecting:
    1
ok
*****
*****
**
2 items had failures:
  2 of  2 in fact
  1 of  2 in fact.fact
4 tests in 2 items.
1 passed and 1 failed.
***Test Failed*** 1 failure.
```

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## pdb debugger commands

---



### pdb Module

- It is a module with a set of utilities for debugging of Python programs.
- pdb internally uses bdb (basic debugger) and cmd (command interpreters) modules.
- pdb runs purely in the command line.
- Pdb supports setting breakpoints, stepping through code, source code listing, viewing stack traces.



# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## pdb debugger commands

---



### pdb Module

- Pdb debugger can be invoked in two ways
  1. Command Line  
`python -m pdb fileName.py`
  2. Importing `pdb` module and call `pdb.set_trace()`

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## pdb debugger commands



- **Command Line**

1. Create a Python Script

```
def fact(n):  
    f = 1  
    for i in range(1,n+1):  
        print (i)  
        f = f * i  
    return f  
  
print("Factorial of 5 =",fact(5))
```

← pdbExample.py

2. Open Command Prompt

3. Change to the directory location where python is installed

`cd C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311`

3. Type the below command

`python -m pdb pdbExample.py`

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands



```
Command Prompt - python · X + v
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SOWMYA SHREE P>cd C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311

C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311>python -m pdb pdbExample.py
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(1)<module>()
-> def fact(n):
(Pdb) |
```

Now, type help in front of the debugger prompt to know more about any command.

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## pdb debugger commands



```
Command Prompt - python - X + v
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SOWMYA SHREE P>cd C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311

C:\Users\SOWMYA SHREE P\AppData\Local\Programs\Python\Python311>python -m pdb pdbExample.py
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(1)<module>()
-> def fact(n):
(Pdb) --KeyboardInterrupt--
(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) --KeyboardInterrupt--
(Pdb) |
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands



- **list** command - lists entire code with -> symbol to the left of a line at which program has halted.

```
(Pdb) list
1  -> def fact(n):
2      f = 1
3      for i in range(1,n+1):
4          print (i)
5          f = f * i
6      return f
7
8      print("Factorial of 5 =",fact(5))
[EOF]
(Pdb) |
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands

---



- **step** command – move line by line, will cause a program to stop within a function

```
(Pdb) step
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(8)<module>()
-> print("Factorial of 5 =",fact(5))
(Pdb) step
--Call--
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(1)fact()
-> def fact(n):
(Pdb) step
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(2)fact()
-> f = 1
(Pdb) |
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands



- **next** command - move line by line, executes a called function and stops after it.

```
(Pdb) next
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(3)fact()
-> for i in range(1,n+1):
(Pdb) next
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(4)fact()
-> print (i)
(Pdb) next
1
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(5)fact()
-> f = f * i
(Pdb) next
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(3)fact()
-> for i in range(1,n+1):
(Pdb) next
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(4)fact()
-> print (i)
(Pdb) next
2
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(5)fact()
-> f = f * i
(Pdb) |
```

# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## pdb debugger commands



- **break** command – set breakpoints within a program. Line number must be given.

```
(Pdb) break 4
Breakpoint 1 at c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py:4
(Pdb) list
 1 -> def fact(n):
 2     f = 1
 3     for i in range(1,n+1):
 4 B         print (i)
 5         f = f * i
 6     return f
 7
 8     print("Factorial of 5 =",fact(5))
[EOF]
(Pdb) |
```

- **continue** command – program execution will proceed till it encounters a breakpoint

```
(Pdb) continue
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(4)fact()
-> print (i)
(Pdb) |
```



## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands



- **break** command – Display all break points using break command without line number

```
(Pdb) break
Num Type      Disp Enb   Where
1  breakpoint keep yes    at c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py:4
    breakpoint already hit 1 time
(Pdb) |
```

- **continue** command – program execution will proceed till it encounters a breakpoint

```
(Pdb) continue
> c:\users\sowmya shree p\appdata\local\programs\python\python311\pdbexample.py(4)fact()
-> print (i)
(Pdb) |
```

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### pdb debugger commands



- Using `pdb.set_trace()`

The Pdb debugger can be used from within Python script also

#### 1. import pdb

```
import pdb
def fact(n):
    f = 1
    for i in range(1,n+1):
        pdb.set_trace()
        print (i)
        f = f * i
    return f

print("Factorial of 5 =",fact(5))
```

← pdbExample.py

#### 2. Call `set_trace` function

The behavior of the debugger will be exactly the same as we find it in a command line environment.

## PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

### Testing – Pytest, Function testing with Doctest, pdb debugger commands

---



#### Summary

- **pytest** – a testing framework in Python, helps to write tests from simple unit tests to complex functional tests.
- **doctest** - a module that verifies whether the code work as intended. It allows generation of tests based on output from the standard Python interpreter shell.
- **pdb** - a module with a set of utilities for debugging of Python programs.



**THANK YOU**

---

Department of Computer Science and Engineering

Prof. Sindhu R Pai – [sindhurpai@pes.edu](mailto:sindhurpai@pes.edu)

Prof. Sowmya Shree P