PCPS

Unit - 3

What's Recursion?

- In simple words, its a function calling itself.
- It reduces length of code & helps solve complex problems

Note:

- Recursive function must have a terminating condition & each call must lead to this condition

```
def fact(n):
    if n==0:
        res = 1
    else:
        res=n*fact(n-1)
    return res
print(fact(5))
```

Terminating Condition

fact(5)
res = 5 x fact(4)
→ 4 x fact(3)
→ 3 x fact(2)
→ 2 x fact(1)
→ 1

5! = 120

```
def gcd(m,n):
    if m==n:
        res =m
    elif m>n:
        res = gcd(m-n,n)
    else:
        res = gcd(m,n-m)
    return res
print(gcd(78,65))
```

78, 65
→ 13, 65
→ 13, 52
→ 13, 39
→ 13, 26
→ 13, 13

GCD = 13

Now Try finding terminating condition & output using the above logic for these programs

```
def add(m,n):
    if n==0:
        return m
    return add(m,n-1)+1
print(add(20,10))
```

```
def mul(m,n):
    if m<n:
        return mul(n,m)
    elif(n!=0):
        return (m+mul(m,n-1))
    else:
        return 0
print(mul(10,5))
```

```
def sub(m,n):
    if n==0:
        return m
    return sub(m,n-1)-1
print(sub(20,10))
```

```
def div(m,n):
    if m<n:
        return 0
    else:
        return 1 + div(m-n,n)
print(div(20,5))
```

What's the difference b/w Recursion & Iteration?

✓ Better ☺

Recursion	Iteration
• Function calls itself	Set of program statements executed repeatedly
• Implemented using functions	Implemented using loops
• Termination defined within recursion function	Termination defined in definition of loop
• If no termination, leads to ∞ recursion	If no termination, leads to ∞ loop
• Slower	Faster
• Uses more memory	Uses less memory

What's a callback?

- Function that is passed to another function as argument

```
def thank(name):
    return f"Thank you {name}"

def greet(name, callback_func):
    greet_msg = callback_func(name)
    print(greet_msg)

greet('Auto Anna',thank)
```

→ Call back function

→ Thank you Auto Anna

→ Higher-order function

```
def multiply(x):
    return num_list[0]*num_list[1]

def compute(func,x):
    return func(x)

num_list=[2,4]
product = compute(multiply,num_list)
print(product)
```

→ 8

What's a closure?

(Not your relationship 😉)

- It is a nested function with access to a free variable from an enclosing function that has finished execution.

```
def outer_func():
    message = 'hi'

    def inner_func():
        print(message)
    return inner_func()
outer_func()
```

Outer function defined

Inner function is defined

Now inner function has access to a free variable from the outer function

```
hi
```

Now,

```
def outer_func():
    message = 'hi'

    def inner_func():
        print(message)
    return inner_func

new_func = outer_func()
print(new_func)
print(new_func.__name__)
new_func()
```

```
<function outer_func.<locals>.inner_func at 0x000002167A215300>
inner_func
hi
```

Inner Function

Assigning a variable with function.

So, we can treat a function as an object

Says that variable is a function

```
def outer_func(msg):
    def inner(m=msg):
        print(m,'world')
    return inner

different= outer_func(msg = 'Hello')
different()
```

```
Hello world
```

Takes the free variable from outer function and uses it in the inner function

What are Decorators?

- They wrap a function & modify its behavior in one way or the other w/o changing source code of the function being decorated

```
def decorator_function(original_function):
    def wrapper_function():
        print('1st output')
        original_function()
        print('3rd output')
    return wrapper_function

def display():
    print('2nd output')

decorated_display = decorator_function(display)
decorated_display()
```

Code of Wrapper Function

decorated function

```
1st output
2nd output
3rd output
```

same code in diff. format

```
def decorator_function(original_function):
    def wrapper_function():
        print('1st output')
        original_function()
        print('3rd output')
    return wrapper_function

@decorator_function
def display():
    print('2nd output')

display()
```

Mention decorator function

Mention decorated function

```
def decorator_x(func):
    def inner_func():
        print("X"*20)
        func()
        print("x"*20)
    return inner_func

def decorator_y(func):
    def inner_func():
        print("Y"*20)
        func()
        print("y"*20)
    return inner_func

@decorator_x
@decorator_y
def display():
    print('The middle part')

display()
```

```
XXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYYY
The middle part
yyyyyyyyyyyyyyyyyyy
xxxxxxxxxxxxxxxxxxx
```

Multiple decorators

What are Generators?

- Its a function that returns an iterator that produces a sequence of values when iterated over faster than normal iteration
- Here yield is used in place of return

```
def z(nums):
    for i in nums:
        yield(i**2)

my_nums = z([1,2,3,4,5])

print(next(my_nums))
print(next(my_nums))
print(next(my_nums))
print(next(my_nums))
print(next(my_nums))
```

Generator Function

Generator Object

Will print Generator object one by one

```
1
4
9
16
25
```

Same Output

```
def z(nums):
    for i in nums:
        yield(i**2)

my_nums = z([1,2,3,4,5])

for num in my_nums:
    print(num)
```

Using for loop instead of writing multiple times

What's a Generator Expression then?

- It is another way to write the entire generator function in 1 expression
- It is similar to list comprehension, but instead of storing the

```
z = (i**2 for i in range(1,6))
for i in z:
    print(i)
```

We expressed it completely into 1 single line & printed required output.

Gives same output

What's the difference b/w yield & return then?

YIELD	RETURN
• Returns a value & passes the execution while maintaining the internal states	Returns a value & terminates the execution of the function
• Converts regular function into generator function	Returns result to caller statement
• Used when generator returns an intermediate result to caller	Used when function is ready to send value
• Code after yield execute after next function call	Code after return wont execute
• Can run multiple times	Runs only single time

NOTE:

```
def z(nums):
    x,y=0,1
    for i in range(nums):
        x,y = y,x+y
        yield x
def square(nums):
    for num in nums:
        yield num**2

print(square(z(5)))
```

Generator Function for fibonacci series

Square Generator Function

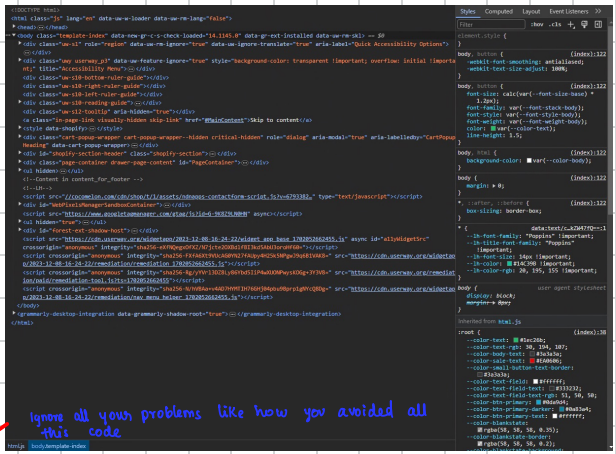
Here we have printed in the normal way as we use with return.

```
<generator object square at 0x000001D15B0F05F0>
```

We have to use print(next()) for generator functions to give desired output

All the coding part is fine. How do I make a user interface?

- We use Tkinter GUI as a simple interface



This is much easier to use

Rather than trying to understanding all of this

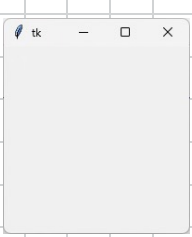
- Tkinter is built into python standard library

- Tkinter can:
  - Create Windows
  - Build GUI for desktop apps
  - Add GUI to command line program
  - Create custom widgets
  - Prototype GUI

What's the simplest form of this GUI?

```
import tkinter
root = tkinter.Tk()
root.mainloop()
```

→ Creates window



↳ loop continuously until we close the window

How do add title & geometry to Window ?

```
import tkinter
root = tkinter.Tk()
root.title('Tkinter Demonstration')
root.geometry('500x500')
root.mainloop()
```

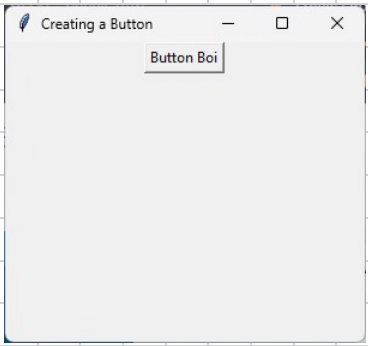
What should I do with an empty screen!? I want some functionalities!

- After creating window, we add some elements called 'widgets' to make it more interactive
- When creating one, we must pass its parent as a parameter. Except 'root', every thing else will have parent.

```
from tkinter import *
root = Tk()
root.title('Creating a Button')
root.geometry('300x250')
b=Button(root,text = 'Button Boi')
b.pack()
root.mainloop()
```

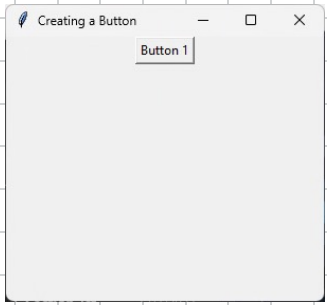
packing it into the window

Description of Button



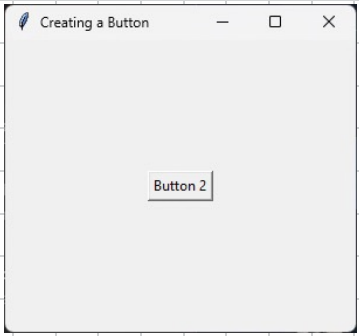
What are all the ways to display a button in the window?

```
from tkinter import *
root = Tk()
root.title('Creating a Button')
root.geometry('300x250')
a=Button(root,text = 'Button 1')
a.pack()

root.mainloop()
```



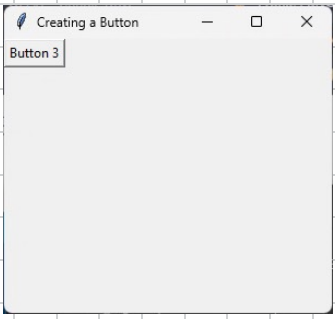
Packs widgets in rows/columns

```
from tkinter import *
root = Tk()
root.title('Creating a Button')
root.geometry('300x250')
b=Button(root,text = 'Button 2')
b.place(relx = 0.5, rely = 0.5, anchor = CENTER)

root.mainloop()
```



explicitly set the position & size of a window, either in absolute terms (or) relative to another window

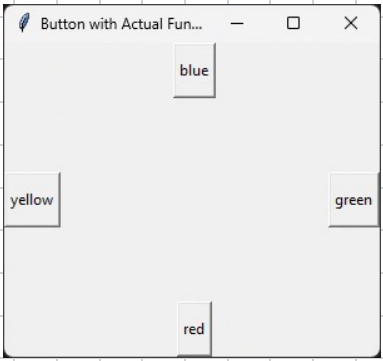
```
from tkinter import *
root = Tk()
root.title('Creating a Button')
root.geometry('300x250')
c=Button(root,text = 'Button 3')
c.grid()

root.mainloop()
```

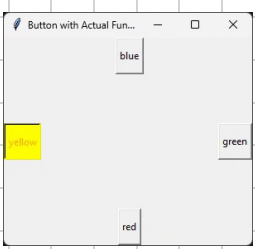


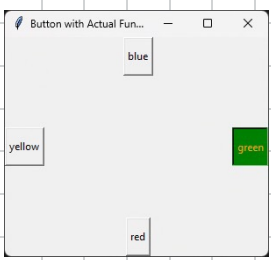
Puts widget in 2D table. Master widget is split into a no. of rows & columns & each cell in resulting table can hold a widget

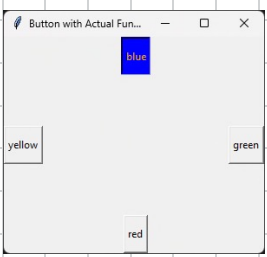
How do I use Button widget?

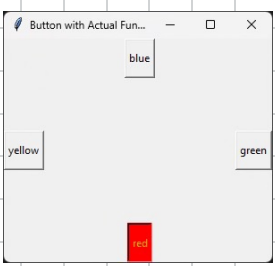
```
import tkinter
from tkinter import *
from tkinter import messagebox

win = Tk()
win.title('Button with Actual Functionality')
win.geometry('300x250')

def green():
    messagebox.showinfo('Message','Green Button clicked')
def red():
    messagebox.showinfo('Message','Red Button Clicked')
def blue():
    messagebox.showinfo('Message','Blue Button Clicked')
def yellow():
    messagebox.showinfo('Message','Yellow Button Clicked')

a=Button(win,text='yellow',command=yellow,activeforeground='orange',activebackground='yellow',pady=10)
b=Button(win,text='green',command=green,activeforeground='orange',activebackground='green',pady=10)
c=Button(win,text='blue',command=blue,activeforeground='orange',activebackground='blue',pady=10)
d=Button(win,text='red',command=red,activeforeground='orange',activebackground='red',pady=10)
a.pack(side=LEFT)
b.pack(side=RIGHT)
c.pack(side=TOP)
d.pack(side=BOTTOM)
win.mainloop()
```

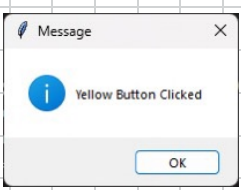


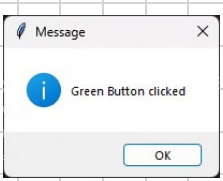


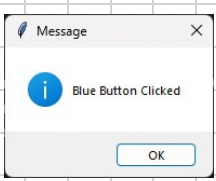


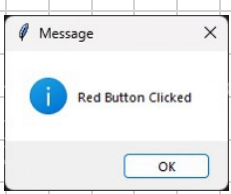












What are Button Widget Options?

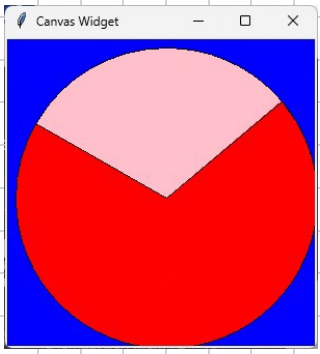
- activebackground
  - ↳ background when mouse hovers over button
- activeforeground
  - ↳ font color when mouse hovers over button
- bd
  - ↳ border width
- bg
  - ↳ background color of button
- fg
  - ↳ foreground color of button
- height
  - ↳ height of button
- justify
  - ↳ LEFT, RIGHT, CENTER
- underline
  - ↳ underline text of button
- width
  - ↳ width of button

How do I use Canvas widget?

```
from tkinter import *
win = Tk()
win.title('Canvas Widget')
win.geometry('300x300')

cv = Canvas(win,bg = 'blue',height='300')

coord= 10,10,300,300
arc1 = cv.create_arc(coord,start=0,extent=150,fill='pink')
arc2 = cv.create_arc(coord,start=150,extent = 250,fill='red')
cv.pack()
win.mainloop()
```

made bg color as blue

Creating 2 arcs




```
from tkinter import *
win = Tk()
cv = Canvas(win,height=225,width=225)
filename = PhotoImage(file="rick.png")

image = cv.create_image(1,1,anchor=NW,image=filename)

cv.pack()
win.mainloop()
```

Positions

Make sure to use only GIF, PGM, PPM and PNG formats

What are Canvas Widget Options?

- bd
  - ↳ border width
- bg
  - ↳ background color
- cursor
  - ↳ to use arrow, dot, or circle
- height
  - ↳ height of canvas
- x cursor command
  - ↳ horizontal Scrollbar
- y cursor command
  - ↳ vertical Scrollbar
- confine
  - ↳ non-scrollable outside scroll region

How do I use Checkbutton widget?

```
from tkinter import *
win = Tk()
win.geometry('300x300')

w=Label(win,text='Select Hobbies:',fg = 'Blue',font='100')
w.pack()
Checkbutton1= IntVar()
Checkbutton2= IntVar()
Checkbutton3= IntVar()
cb1=Checkbutton(win, text="Painting",variable = Checkbutton1,onvalue = 1,offvalue = 0,height = 2,width = 10)
cb2=Checkbutton(win, text = "Dancing", variable = Checkbutton2,onvalue = 1,offvalue = 0,height = 2,width = 10)
cb3=Checkbutton(win, text = "Cooking", variable = Checkbutton3,onvalue = 1,offvalue = 0,height = 2,width = 10)

cb1.pack()
cb2.pack()
cb3.pack()

mainloop()
```

Holds integer data passed to checkbutton widget

When checked, stores as 1

When not, stores as 0

What are Checkbox Widget Options?

- bd
  - Border width
- bg
  - Background color
- bitmap
  - Display image in button
- command
  - Function to be called when checked
- height
  - height of widget
- image
  - Display image in button
- justify
  - LEFT, RIGHT, CENTER
- padx
  - Space to leave to left & right of checkbutton & text. Default = 1
- pady
  - Space to leave to top & bottom of checkbutton & text. Default = 1

What are Checkbox Widget Functions?

- deselect()
  - Turn off check button
- flash()
  - Check button flashed b/w active & normal colors
- invoke()
  - invoke method associated with checkbutton
- select()
  - To turn on checkbutton
- toggle()
  - Toggle b/w different checkbuttons

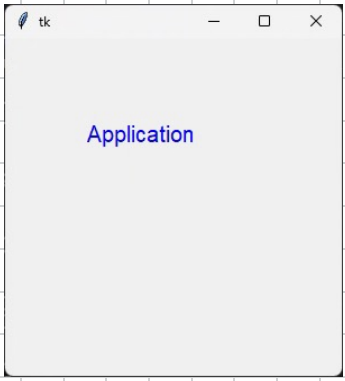
How do I use label widget?

```
from tkinter import *
win = Tk()
win.geometry('300x300')

w = Label(win,text = 'Application',fg='blue',font='100').place(x=70,y=70)
mainloop()
```

Parent

location in window

Options



What are Label Widget Options?

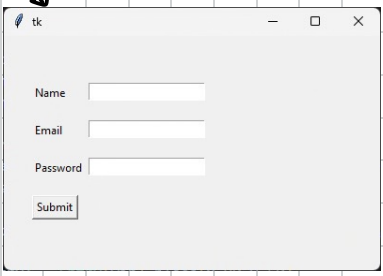
- anchor
  - ↳ to control position of widget
- bg
  - ↳ Background color
- bitmap
  - ↳ Display image in button
- cursor
  - ↳ to use arrow, dot, or circle
- height
  - ↳ height of widget
- image
  - ↳ Display image in button
- justify
  - ↳ LEFT, RIGHT, CENTER
- padx
  - ↳ Space to leave to left & right of checkbutton & text. Default = 1
- pady
  - ↳ Space to leave to top & bottom of checkbutton & text. Default = 1

How do I use Entry widget?

```
from tkinter import *
win = Tk()
win.geometry('400x250')

name = Label(win,text='Name').place(x=30,y=50)
email = Label(win,text='Email').place(x=30,y=90)
password = Label(win,text='Password').place(x=30,y=130)

subbtn = Button(win,text = 'Submit',activebackground='red',activeforeground='blue').place(x=30,y=170)

entry1 = Entry(win).place(x=90,y=50)
entry2 = Entry(win).place(x=90,y=90)
entry3 = Entry(win).place(x=90,y=130)

win.mainloop()
```

parent

Location



What are Entry Widget Options?

- bg
  - ↳ Background color
- font
  - ↳ font used for the text
- fg
  - ↳ color to render text
- relief
  - ↳ can be FLAT, SUNKEN, RIGID, RAISED, GROOVE
- show
  - ↳ To show text while making an entry. Ex: for Password set Show = "*"
- textvariable
  - ↳ To retrieve the current text from your entry widget

What are Checkbox Widget Functions?

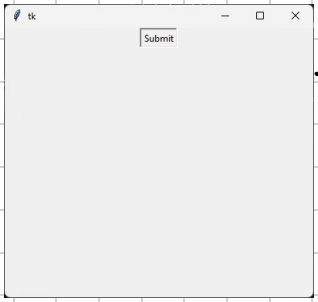
- get()
  - ↳ Returns entry's current text as string
- delete()
  - ↳ Deletes character from widget
- insert (index, name):
  - ↳ Inserts string 'name' before the character at the given index

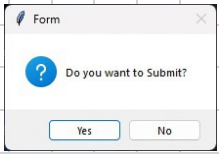
How do I use a dialog box?

```
from tkinter import *
from tkinter import messagebox
win = Tk()

def Submit():
    messagebox.askquestion("Form","Do you want to Submit?")

win.geometry('400x350')

b=Button(win,text='Submit',command= Submit)
b.pack()

win.mainloop()
```

message box

executing the function defined





What are the other functions we can use?

- showinfo()
  - ↳ Display info
- showwarning()
  - ↳ Display Warning
- showerror()
  - ↳ Display Error
- askquestion()
  - ↳ Dialog box that asks YES or NO
- askokcancel()
  - ↳ Dialog box that asks OK or CANCEL
- askretrycancel()
  - ↳ Dialog box that asks RETRY or CANCEL
- askyesnocancel()
  - ↳ Dialog box that asks YES or NO or CANCEL

What are Modules?

- It is a file containing code that can be reused & organises it into modules logically
- The main reason we use it is to reduce the size of code & similar stuff can be grouped easily

- They are 2 types:
  - Built-In
    - ↳ Python's standard libraries comes with lot of modules bundled together
  - User Defined
    - ↳ Any file with .py extension that consists of code is a module

Let this be the module

```
print('Module has been imported')

def find_index(to_search,target):
    for i,value in enumerate(to_search):
        if value == target:
            return i
    return -1

test = 'Test Function'
```

imported module and renamed it

```
import module1 as m1
a = ['math','phy','chem','cs','ece','eee','math','chem','eee']

print(m1.find_index(a,'eee'))
print(m1.test)
```

```
Module has been imported
5
Test Function
```

Used a function from module

We could also import only a few of these functions from the module.

Instead of use `import module1 as m1`

`from module1 import find_index as fi, test`

Where does the program know where to search from?

- The program searches a few directories.

```
import sys
print(sys.path)
```

→ Gives a list of all the directories it searches from

```
['c:\\Users\\hites\\Documents\\python notes progress', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2032.0_x64__qbz5n2kfra8p0\\python311.zip', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2032.0_x64__qbz5n2kfra8p0\\DLLs', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2032.0_x64__qbz5n2kfra8p0\\Lib', 'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.11_3.11.2032.0_x64__qbz5n2kfra8p0', 'c:\\Users\\hites\\Documents\\python notes progress\\myenv', 'c:\\Users\\hites\\Documents\\python notes progress\\myenv\\Lib\\site-packages']
```

→ The directory that the program is in

What if I want to add a custom directory?

- You can treat sys.path as a list and append the directory of choice

sys.path.append ('File location')

NOTE: This is temporary and will be undone when closed.

What are the importing mechanisms?

```
print('file1__name__=%s'%__name__)

if __name__ == '__main__':
    print('file1 is execute')
else:
    print('file1 is imported')

def func(a,b):
    '''used to add the 2 values'''
    return a+b
```

→ A special variable that when executed in main program, value is "__main__"

```
file1__name__=__main__
file1 is execute
```

```
import module1 as m1

print ("file2 __name__ = %s" %__name__)

if __name__ == '__main__':
    print('file2 is exectuted directly')
else:
    print('file2 is imported')


print(m1.func.__doc__)
help(m1)
```

→ But when imported, it is set to module's name

→ Used to show the documentation related to it

```
file1__name__=module1
file1 is imported
file2 __name__ = __main__
file2 is exectuted directly
used to add the 2 values
Help on module module1:

NAME
    module1

FUNCTIONS
    func(a, b)
        used to add the 2 values

FILE
    c:\users\hites\documents\python notes progress\module1.py
```

How do I test my code?

- There's 3 ways:
  1) pytest :
     A testing framework that helps write tests from simple unit tests to complex functional steps
  2) doctest :
     A module that verifies whether code works as intended
  3) pdb :
     A module with a set of utilities for debugging of Python prog.

Why use PyTest?

1) Free & Open Source
2) Simple Syntax
3) Runs multiple tests in parallel, which reduces execution time of test suite
4) Can automatically detect test file & functions even if not mentioned
5) Allows to skip a subset of test during execution
6) Allows to run a subset of test suite

- Doesn't need API
- Provides useful plugins
- Can be written as function or method
- Gives useful failure info w/o debuggers
- Can be used to run doc tests & unit tests

How do I test using pytest

1) Create new directory 'automation' & navigate into it in command line
2) Create file pytestexample.py
3) Run using command
   pytest pytestexample.py

```python
import math

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

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

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

```
================================================================================ 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 0.12s ================================================================================
PS C:\Users\hites\Documents\python notes progress\automation>
```

How do I use doctest

```
from doctest import testmod
def fact(n):
    '''
    >>> fact(5)
    120
    >>> fact(3)
    6
    '''
    if n==0:
        res = 1
    else:
        res = n*fact(n-1)
    return res
testmod(name='fact',verbose=True)
```

import this

Write
>>> funcname(arg)
Expected Output
and
then write
logic

Call doctest

```
Trying:
    fact(5)
Expecting:
    120
ok
Trying:
    fact(3)
Expecting:
    6
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.
```

```
from doctest import testmod

def fact(n):
    '''
    >>> fact(5)
    120
    >>> fact(1)
    1
    '''
    if n==0:
        res = 1
    else:
        res = fact(n-1)
    return res
testmod(name='fact',verbose=True)
```

```
Trying:
    fact(5)
Expecting:
    120
**********************************************************************
File "C:\Users\hites\Documents\python notes progress\automation\pytestexample.py", line 9, in fact.fact
Failed example:
    fact(5)
Expected:
    120
Got:
    1
Trying:
    fact(1)
Expecting:
    1
ok
1 items had no tests:
    fact
**********************************************************************
1 items had failures:
   1 of   2 in fact.fact
2 tests in 2 items.
1 passed and 1 failed.
***Test Failed*** 1 failures.
PS C:\Users\hites\Documents\python notes progress\automation>
```

If verbose is false,
output will be shown in case of failure only
& not in case of success

How do I use pdb debugger?

- There are 2 ways:
  1) Command line
     python -m pdb FileName.py
  2) import pdb
     & call 'pdb.set_trace()'

Open Command Prompt
Open directory where file is there
Type 'python -m pdb Filename.py'

Gives same interface

```
-> def fact(n):
(Pdb)
```

Should get this

```
(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)
```

use list
will show entire code & -> represents where code is halted

```
-> 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)
```

help shows all the commands

Similarly use **step**, **next**, **break no.**, **continue**, **break**

- **step**: move line by line; will cause program to stop within a function
- **next**: moves line by line; executes called func & stops after it
- **break no.**: set break points within program; no. = line number; must input a number
- **continue**: Program is executed till it finds breakpoint
- **break**: Displays all breakpoints using break point