Subroutine (continue)

01204111 Section 1

Task: *print triangle2*

- Writing the printTriangle2(x) which passing x as integer and printing the triangle.

INPUT:

```
Input size: 3
```

OUTPUT:

```
  *
 **
***
 **
  *
```

INPUT:

```
Input size: 5
```

OUTPUT:

```
    *
   **
  ***
 ****
*****
 ****
  ***
   **
    *
```

INPUT:

```
Input size: 6
```

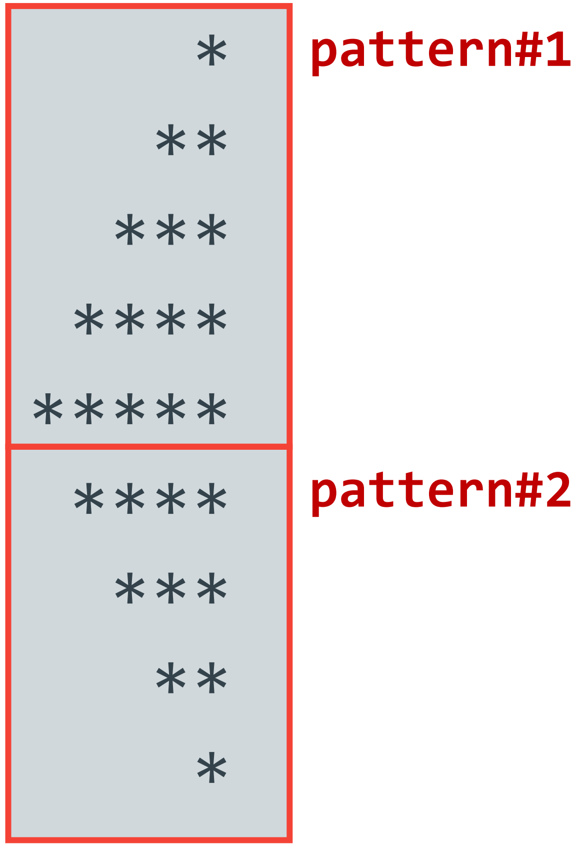
OUTPUT:

```
     *
    **
   ***
  ****
 *****
******
 *****
  ****
   ***
    **
     *
```

Task: *print triangle2*

- Steps
  - Separate to Two patterns
  - Find Number of row in pattern#1 and #2
  - Print spacebars and stars per row

```
def printTriangle2(x) :
     pattern1(x)
     pattern2(x)
```



```
    *
   **
  ***
 ****
*****
 ****
  ***
   **
    *
```

pattern#1

pattern#2

Task: *print triangle2 Ver.1*

```
def printTriangle2(x) :
    pattern1(x)
    pattern2(x)

def pattern1(x) :
    maxRow = findMaxRow1(x)
    row = 0
    while row < maxRow :
        printSpaces1(row,x)
        printStars1(row)
        print()
        row += 1
```

```
def findMaxRow1(x) :
    return x
def printSpaces1(row,x) :
    nbSpaces = x - row - 1
    i = 0
    while i < nbSpaces :
        print(' ',end='')
        i += 1
def printStars1(row) :
    nbStars = row + 1
    i = 0
    while i < nbStars :
        print('*',end='')
        i += 1
```

Task: *print triangle2 Ver.1*

```
def pattern2(x) :
    maxRow = findMaxRow2(x)
    row = 0
    while row < maxRow :
        printSpaces2(row)
        printStars2(row,x)
        print()
        row += 1
```

```
def findMaxRow2(x) :
    return x-1
def printSpaces2(row) :
    nbSpaces = row + 1
    i = 0
    while i < nbSpaces :
        print(' ',end='')
        i += 1
def printStars2(row,x) :
    nbStars = x - row - 1
    i = 0
    while i < nbStars :
        print('*',end='')
        i += 1
```

Task: *print triangle2 Ver.2*

```
def printTriangle2(x) :
      pattern1(x)
      pattern2(x)

def pattern1(x) :
      maxRow = findMaxRow1(x)
      row = 0
      while row < maxRow :
            nbSpaces = findSpaces1(row,x)
            printSpaces(nbSpaces)
            nbStars = findStars1(row)
            printStars(nbStars)
            print()
            row += 1
```

```
def findMaxRow1(x) :
      return x
def findSpaces1(row,x) :
      return x - row - 1
def findStars1(row) :
      return row + 1
def printSpaces(nbSpaces) :
      i = 0
      while i < nbSpaces :
            print(' ',end='')
            i += 1
def printStars(nbStars) :
      i = 0
      while i < nbStars :
            print('*',end='')
            i += 1
```

Task: *print triangle2 Ver.2*

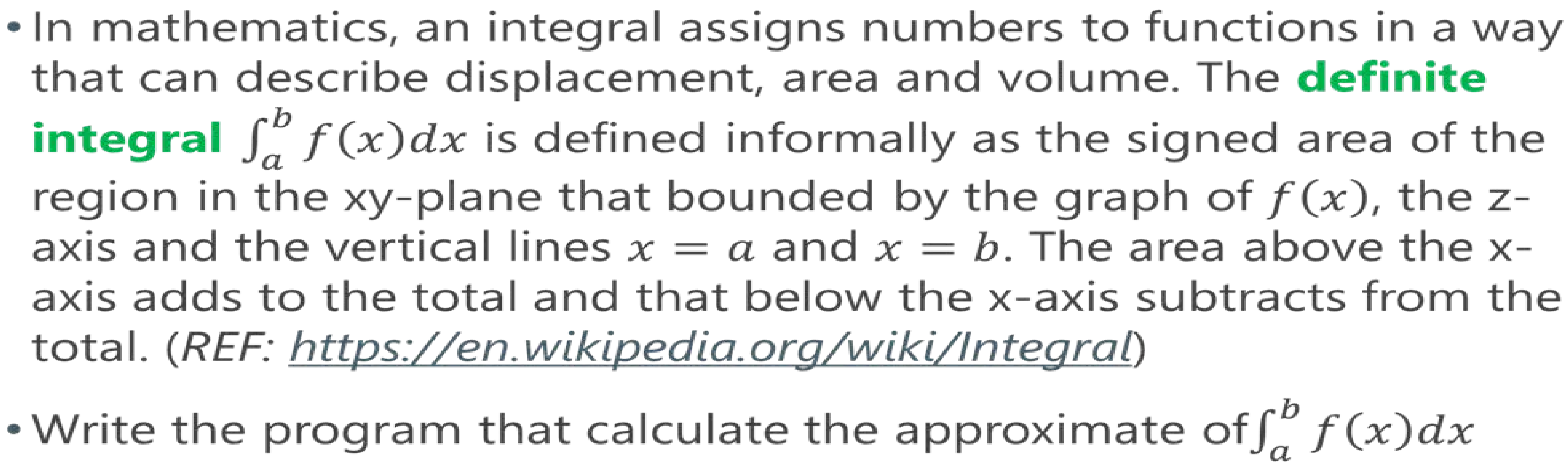
```
def pattern2(x) :
    maxRow = findMaxRow2(x)
    row = 0
    while row < maxRow :
        nbSpaces = findSpaces2(row)
        printSpaces(nbSpaces)
        nbStars = findStars2(row,x)
        printStars(nbStars)
        print()
        row += 1
```

```
def findMaxRow2(x) :
    return x - 1
def findSpaces2(row) :
    return row + 1
def findStars2(row,x) :
    return x - row - 1
```

Task: *print triangle2*

```
# printSpaces(nbSpaces)  --> printNTimes(nbSpaces,' ')
# printStars(nbStars)    --> printNTimes(nbStars,'*')
def printNTimes(nb,character) :
      i = 0
      while i < nb :
            print(character,end='')
            i += 1
```

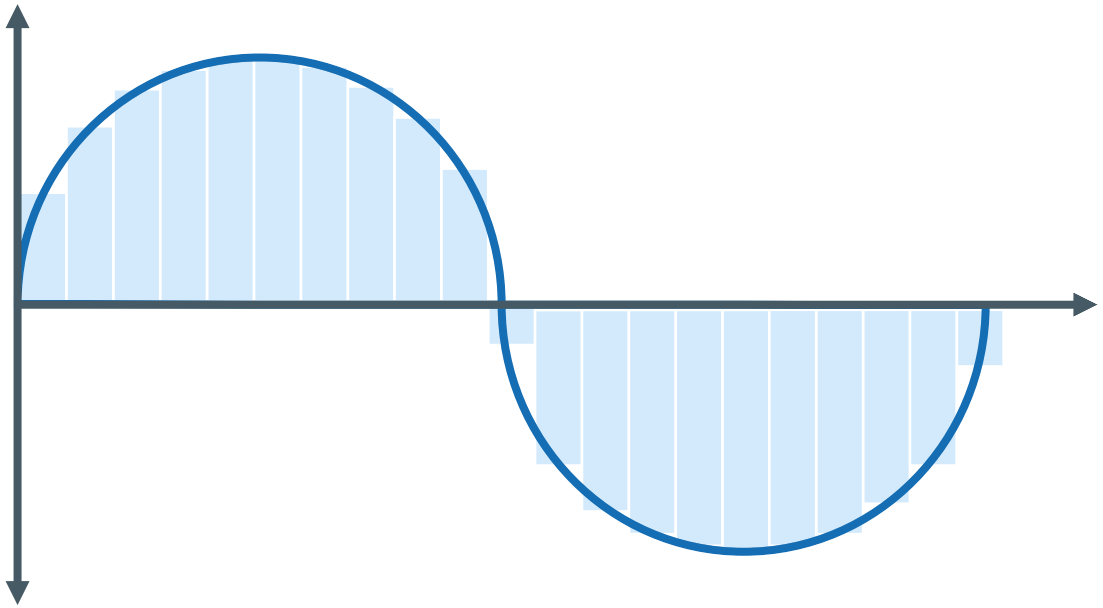
Task: *Definite Integral*



- In mathematics, an integral assigns numbers to functions in a way that can describe displacement, area and volume. The **definite integral** $\int_a^b f(x)dx$ is defined informally as the signed area of the region in the xy-plane that bounded by the graph of $f(x)$, the z-axis and the vertical lines $x = a$ and $x = b$. The area above the x-axis adds to the total and that below the x-axis subtracts from the total. (*REF: https://en.wikipedia.org/wiki/Integral*)
- Write the program that calculate the approximate of$\int_a^b f(x)dx$

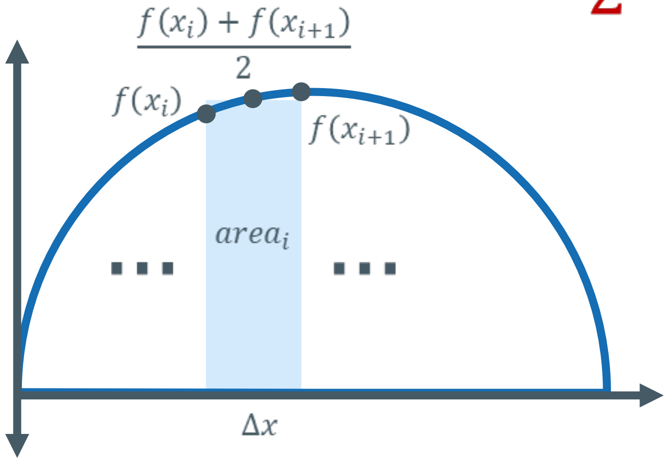
Task: *Definite Integral*

- **Hint:** you can find $\int_a^b f(x)dx$ from summation of small areas between the $f(x)$ and $x = 0$



- You can find each small area from

$$area_i = \Delta x \times (\frac{f(x_i) + f(x_{i+1})}{2})$$



Task: *Definite Integral*

```
import math

def f(x) :
      return math.sin(x)

def integral (a,b,nbOfArea) :
      deltaX = (b-a)/nbOfArea
      sum = 0
      i = a
      while i < b :
            yNow = f(i)
            yNext = f(i+deltaX)
            sum += deltaX * (yNow+yNext)/2
            i += deltaX
      return sum
```

```
a = 0
b = 2*math.pi
nbOfArea = 10000
signedArea = integral(a,b,nbOfArea)
print('Signed Area: %.2f' %(signedArea))
```

OUTPUT:

```
Signed Area: 0.00
```

CHECK

$$\int_0^{2\pi} sin(x)dx = [-\cos(x)]_0^{2\pi}$$

$$= -\cos(2\pi) + \cos(0)$$

$$= -1 + 1 = 0$$

Task: *Unsigned Area*

- Modified the *Definite Integral* 's code to find the total unsigned area between $f(x) \ = cos(x)$ and $g(x) = 0$ from $x = 0$ to $x = 2\pi$
- HINT: The answer must equal 4.00

Task: *Unsigned Area*

```python
import math

def f(x) :
    return math.cos(x)

def integral (a,b,nbOfArea) :
    deltaX = (b-a)/nbOfArea
    sum = 0
    i = a
    while i < b :
        yNow = f(i)
        yNext = f(i+deltaX)
        sum += abs(deltaX * (yNow+yNext)/2)
        i += deltaX
    return sum
```

```python
a = 0
b = 2*math.pi
nbOfArea = 10000
area = integral(a,b,nbOfArea)
print('Area: %.2f' %(area))
```

OUTPUT:

```
Area: 4.00
```

Do these codes produce errors?

```
def concatenate(x,y) :
      return x+y

concatenate('CPE','31')
```

No variable collects returning value

- Successfully compile
- Successfully execute
- Semantic error

```
def printConcatenate(x,y) :
      print(x+y)

x = printConcatenate('CPE','31')
```

Variable collects no value

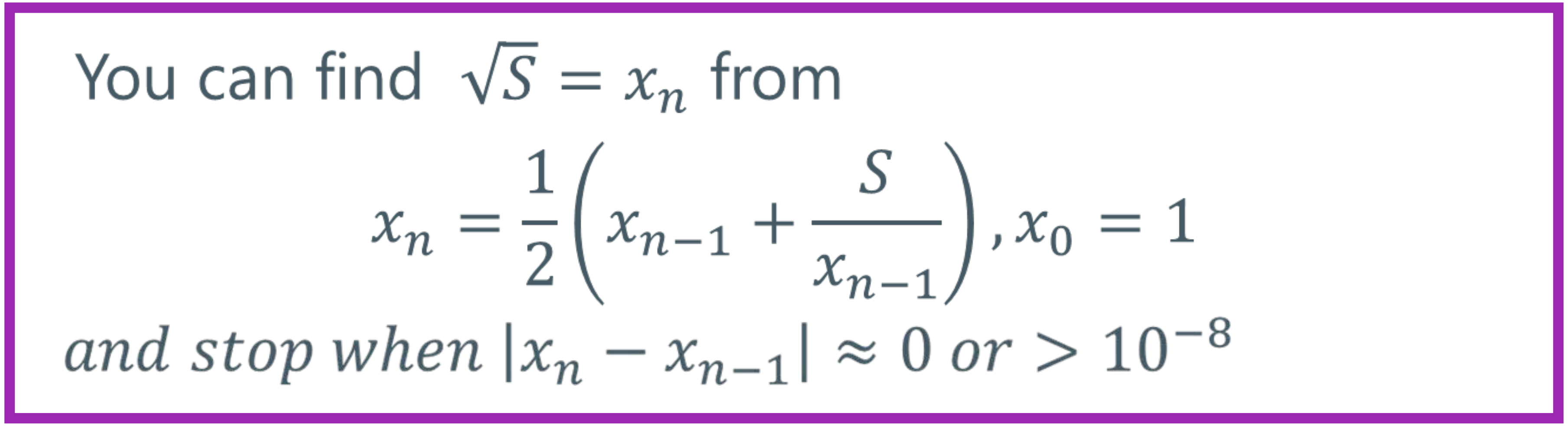
- Successfully compile
- Successfully execute
- Semantic error

```
print(x)
print(type(x))
```

```
None
<class 'NoneType'>
```

Task: *Babylonian Square Root*

- When we want a value of $\sqrt{s}$, we always use $s^{0.5}$ but there are many way to find the square root such as **Babylonian Square Root**. Write the function sqrt(s) that implement from **Babylonian Square Root** algorithm



You can find $\sqrt{S} = x_n$ from

$$x_n = \frac{1}{2}\left(x_{n-1} + \frac{S}{x_{n-1}}\right), x_0 = 1$$

$$and\ stop\ when\ |x_n - x_{n-1}| \approx 0\ or > 10^{-8}$$

Task: *Babylonian Square Root*

```
def absolute(x) :
       if x >= 0 :
             return x
       else :
             return -x
def sqrt(s) :
       x0 = 1; epsilon = 10**(-8)
       xn = (x0+s/x0)/2
       while absolute(xn-x0) >= epsilon :
             x0 = xn
             xn = (x0+s/x0)/2
       return xn
s = float(input('Input S: '))
print('sqrt(%.8f)=%.8f\n%.8f**0.5=%.8f' %(s,sqrt(s),s,s**0.5))
```

Task: *Nilakantha PI*

- **Nilakantha** (an Indian mathematician 1444-1544) propose a formula to calculate **PI** (first 101 terms):

$$3+\frac{4}{2\times 3\times 4}-\frac{4}{4\times 5\times 6}+\frac{4}{6\times 7\times 8}-\ \dots$$

Task: *Nilakantha PI*

- Find the relationship

term	value
0	3
1	$+\frac{4}{2\times3\times4}=\frac{(-1)^{1+1}(4)}{(1\times2)((1\times2)+1)((1\times2)+2)}$
2	$-\frac{4}{4\times5\times6}=\frac{(-1)^{2+1}(4)}{(2\times2)((2\times2)+1)((2\times2)+2)}$
3	$+\frac{4}{6\times7\times8}=\frac{(-1)^{3+1}(4)}{(3\times2)((3\times2)+1)((3\times2)+2)}$
4	$-\frac{4}{8\times9\times10}=\frac{(-1)^{4+1}(4)}{(4\times2)((4\times2)+1)((4\times2)+2)}$

$For\ n \geq 1$

$$x_n = \frac{(-1)^{n+1}(4)}{(n\times2)((n\times2)+1)((n\times2)+2)}$$

$$x_n = \frac{(-1)^{n+1}(4)}{4n(2n^2+3n+1)}$$

$$x_n = \frac{(-1)^{n+1}}{n(2n^2+3n+1)}$$

Task: *Nilakantha PI*

```
def pi() :
    val = 3
    n = 1
    while n <= 101:
        val += ((-1)**(n+1))/(n*(2*(n**2) + 3*n + 1))
        n += 1
    return val
print('Nilakantha PI is %f' %pi())
```

Built-in Functions

- Python provides many mathematical functions
- Some common functions are:

Expression	Evaluated to
abs(x)	$\lvert x \rvert$
pow(x,y)	x^y

max(x,y)
max(x,y,z)
	Find the largest of x,y
	Find the largest of x,y,z

min(x,y)
min(x,y,z)
	Find the smallest of x,y
	Find the smallest of x,y,z

range(x)
range(x,y)
range(x,y,z)
	$0, 1, 2, \ldots, x-1$
	$x, x+1, x+2, \ldots, y-1$
	$x, x+z, x+2z, \ldots, a \mid a < y, a\%z == x$

range(5) = [0,1,2,3,4]
range(5,10) = [5,6,7,8,9]
range(5,13,2) = [5,7,9,11]

```
Python2.7:
   range() return list type
Python3.6:
   range() return range type
```

Math Library

- In addition, Python provides many mathematical functions and constants in the **math** library

Expression	Evaluated to
math.floor(x)	$\lfloor x \rfloor$
math.ceil(x)	$\lceil x \rceil$
math.fabs(x)	$\lvert x \rvert$
math.pow(x,y)	x^y
math.max(x,y)	$\max(x, y)$
math.min(x,y)	$min(x, y)$
math.sqrt(x)	$\sqrt{x}$
math.pi	π
math.e	e

Expression	Evaluated to
math.exp(x)	e^x
math.log(x)	$\ln(x)$
math.log10(x)	$\log(x)$
math.log(x,y)	$log_y(x)$
math.sin(x)	$\sin(x); x[rad.]$
math.acos(x)	$cos^{-1}(x); return\ [rad.]$
math.degrees(x)	$x[rad.]; return\ [deg.]$
math.radians(x)	$x[deg.]; return\ [rad.]$
math.factorial(x)	$x!$

Libraries

- In general, a library is a collection of subroutines your program can use
- In Python, a library is a collection of classes, while each class contains several subroutines. The library can be imported using the **import statement** at the top of your program

```
import math
import time
import random
```

Task: *Guessing Number*

- Write a program that random the number 1-10 then the player will guess the number until it correct

HINT: random.randint(lowest,highest)

```
Welcome, guessing 1-10
Guess: 4
LOWER!!
Guess: 2
HIGHER!!
Guess: 3
CORRECT using 3 time(s)
Bye.
```

Task: *Guessing Number*

```python
import random
print('Welcome, guessing 1-10')
guess = -1
ans = random.randint(0,10)
time = 0
while ans != guess :
      guess = int(input('Guess: '))
      time += 1
      getStatus(guess,ans,time)
```

Write your own getStatus(guess,ans,time)

Task: *Guessing Number*

```
def getStatus(guess,ans,time) :
      if guess == ans :
             print('CORRECT using %d time(s)' %time)
             print('Bye.')
             return
      if guess < ans :
             print('HIGHER!!')
      else :
             print('LOWER!!')
```

Task: *Rock Paper Scissors*

- Write a program that interact with user by playing rock paper scissors

```
How many turns?: 5
[rock/paper/scissors]: paper
player wins
[rock/paper/scissors]: rock
pc wins
[rock/paper/scissors]: rock
[rock/paper/scissors]: love
Try again!!
[rock/paper/scissors]: rock
pc wins
[rock/paper/scissors]: scissors
player wins
Player wins 2 time(s) and pc wins 2 times(s)
```

Task: *Rock Paper Scissors*

```
import random
times = int(input('How many turns?: '))
playerWin = 0
pcWin = 0
while times > 0 :
      pcChoose = pcRandom()
      playerChoose = playerInput()
      pcWin,playerWin = scoring(pcChoose,playerChoose,pcWin,playerWin)
      times -= 1
print('player wins %d time(s) and pc wins %d time(s)' %(playerWin,pcWin))
```

Try writing your own pcRandom() , playerInput(), scoring(pcChoose,playerChoose,pcWin,playerWin)

Task: *Rock Paper Scissors*

```
def pcRandom() :
        choose = random.randint(1,3)
        if choose == 1 :
                return 'rock'
        if choose == 2 :
                return 'paper'
        if choose == 3 :
                return 'scissors'
def playerInput() :
        playerChoose = input('player[rock/paper/scissors]: ')
        while playerChoose != 'rock' and playerChoose != 'paper' and 
playerChoose != 'scissors' :
                print('Try again!!')
                playerChoose = input('player[rock/paper/scissors]: ')
        return playerChoose
```

Task: *Rock Paper Scissors*

```
def scoring(pcChoose,playerChoose,pcWin,playerWin) :
    if pcChoose == playerChoose :
        return pcWin,playerWin
    if (pcChoose == 'rock' and playerChoose == 'paper') or
       (pcChoose == 'paper' and playerChoose == 'scissors') or
       (pcChoose == 'scissors' and playerChoose == 'rock') :
        playerWin += 1
        print('player wins')
    else :
        pcWin += 1
        print('pc wins')
    return pcWin,playerWin
```

Overloaded function

- **Overloaded function** is the feature that allows a class to have two or more functions having **same name**, if **their argument lists are different**.

```
max(a,b)
max(a,b,c)
max(a,b,c,d)
max(a,b,c,d,e)
max(a,b,c,d,e,f)
```

`Try writing your own myMax(a,b,c,d,e,f,g) from myMax(a,b)`

Task: *myMax*

```
def myMax(a,b) :
       if a >= b:
             return a
       else :
             return b
```

```
def myMax(a,b,c) :
       return myMax(myMax(a,b),c)
```

```
def myMax(a,b,c,d) :
       return myMax(myMax(a,b,c),d)
```

```
def myMax(a,b,c,d,e) :
       return myMax(myMax(a,b,c,d),e)
```

```
def myMax(a,b,c,d,e,f) :
       return myMax(myMax(a,b,c,d,e),f)
```

```
def myMax(a,b,c,d,e,f,g) :
       return myMax(myMax(a,b,c,d,e,f),g)
```

•

•

•

Making your own library

- You can create your library by writing functions in another file and call by using **import filename** and **filename.functionName(parameters)**

```
import math
print(math.max(4,5))
print(math.pi)
```

```
def max(a,b) :
      if a >= b:
            return a
      else :
            return b
def findPI() :
      ...<Nilakanta PI>
      return ...
pi = findPI()
```

math.py

Task: *myMath*

- Write **myMath** library including these functions :

myMath.sqrt(x)

myMath.max(x,y)

myMath.max(x,y,z)

myMath.pi

myMath.ceiling(x)

myMath.floor(x)

myMath.absolute(x)

Don’t forget to test your library

Task: *myMath*

```
def absolute(x) :
        ...
def sqrt(x) :
        ...
def max(x,y) :
        ...
def max(x,y,z) :
        ...
def findPI() :
        ...
pi = findPI()
def ceiling(x) :
        res = int(x)
        if x – res > 0 :
                return res + 1
        return res
def floor(x) :
        return int(x)
```

myMath.py

```
import myMath
print(myMath.absolute(-10))
print(myMath.sqrt(16))
print(myMath.max(16,43))
print(myMath.max(16,43,20))
print(myMath.pi)
print(myMath.ceiling(3.6))
print(myMath.floor(3.1))
```

test.py

References

- Python Slides (2017) **– Department of Computer Engineering Kasetsart University**
- Think Python **– Allen B. Downey**
- https://automatetheboringstuff.com/chapter3/