3 Days Training on Python3

Day 1 : Module 3

Muhammad Saufy Rohmad

Module 3 (90 minutes)

Objectives
1. Number Guessing Game
2. Recursion
3. Introduction to Structured Analysis

1. Number Guessing Game

- In this chapter we are going to bring everything we have learned so far together to create a simple number guessing game.
- This will involve creating a new Python program, handling user input, using the if statement as well as using looping constructs.
- We will also, use an additional library or module, that is not by default available by default to your program; this will be the random number generator module

1. Number Guessing Game(2)

- Essentially the program logic is
  - The program randomly selects a number between 1 and 10.
  - It will then ask the player to enter their guess
  - It will then check to see if that number is the same as the one the computer randomly generated; if it is then the player has won.
  - If the player’s guess is not the same, then it will check to see if the number is higher or lower than the guess and tell the player.
  - The player will have 4 goes to guess the number correctly; if they don’t guess the number within this number of attempts, then they will be informed that they have lost the game and will be told what the actual number was.

1. Number Guessing Game(3)

- Creating the game – generate random number

  import random

  number_to_guess = random.randint(1,10)

1. Number Guessing Game(4)

- Creating the game – obtain input from user

  guess = int(input('Please guess a number between 1 and 10: '))

1. Number Guessing Game(5)

- Creating the game – check player guest

```
guess = int(input('Please guess a number between 1 and 10: '))

while number_to_guess != guess:

    print('Sorry wrong number')
```

1. Number Guessing Game(6)

- Creating the game – check guest attempt

```
count_number_of_tries = 1
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
    print('Sorry wrong number')
    if count_number_of_tries == 4:
        break
    # TBD ...
    guess = int(input('Please guess again: '))
    count_number_of_tries += 1
```

1. Number Guessing Game(6)

- Creating the game – notify player higher or lower

```
count_number_of_tries = 1
guess = int(input('Please guess a number between 1 and 10:'))
while number_to_guess != guess:
    print('Sorry wrong number')
    if count_number_of_tries == 4:
        break
    elif guess < number_to_guess:
        print('Your guess was lower than the number')
    else:
        print('Your guess was higher than the number')
    guess = int(input('Please guess again: '))
    count_number_of_tries += 1
```

1. Number Guessing Game(6)

- Creating the game – end of game status

```
if number_to_guess == guess:
    print('Well done you won!')
    print('You took', count_number_of_tries ,
          'goes to complete the game')
else:
    print('Sorry - you loose')
    print('The number you needed to guess was',
          number_to_guess)
print('Game Over')
```

2. Recursion

- Recursion is a very powerful way to implement solutions to a certain class of problems.
- This class of problems is one where the overall solution to a problem can be generated by breaking that overall problem down into smaller instances of the same problem.
- The overall result is then generated by combining together the results obtained for the smaller problems.

2. Recursion(2)

- Recursion in python

```
def recursive_function():
    print('calling recursive_function')
    recursive_function()
```

2. Recursion(3)

- Calculating Factorial Recursively

```
def factorial(n):
    if n == 1: # The termination condition
        return 1 # The base case
    else:
        res = n * factorial(n-1) # The recursive call
        return res


print(factorial(5))
```

3. Introduction to Structured Analysis

- The Structured Analysis methods typically employ a process driven approach (with a set of prescribed steps or stages) which in one way or another consider what the inputs and outputs of the system are and how those inputs are transformed into the outputs.
- The steps involved in Structured Analysis identify these functions and will typically iteratively break them down into smaller and smaller functions until an appropriate level of detail has been reached.
- This process is known as Functional Decomposition.

3. Introduction to Structured Analysis(2)

- any program of a significant size will need to be structured such that it can be:
  - understood easily by other developers
  - tested to ensure that it does what is intended
  - maintained as new and existing requirements evolve
  - debugged to resolve unexpected or undesirable behaviour

3. Introduction to Structured Analysis(2)

- Functional Decomposition is one way in which a system can be broken down into its constituent parts.
- This process of breaking down higher level functions into lower level functions helps with:
  - testing the system (functions can be tested in isolation)
  - understanding the system as the organisation of the functions can give meaning to the code, as well as allowing each function to be understood in isolation from the rest of the system
  - maintaining the system as only those functions that need to be changed may be affected by new or modified requirements
  - debugging the system as issues can be isolated to specific functions which can be examined independently of the rest of the application

3. Introduction to Structured Analysis(3)

- Functional Flow
  - Pseudo Code
  - Data Flow Diagram
  - Sequence Diagrams

3. Introduction to Structured Analysis(4)

- Data Flow Diagrams
  - Process
  - Data Flow
  - Data Store/Warehouse
  - Terminator

3. Introduction to Structured Analysis(5)

- Flowchart
  - Terminal
  - Process
  - Decision
  - Input/Output
  - Flow
  - Predefined Process
  - Stored Data
  - Off Page Connector