Lesson 2- Summary:

1. Blocks and Indentations

   A block of code can be defined by an indentation (tab size)
   As long as one or more code instructions are written with an indentation they are part of the same block and context.

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n-1)

print(f(4))
```

2. Conditions:
   Python uses boolean variables to evaluate conditions.
   The boolean values ***True*** and ***False*** are returned when an expression is compared or evaluated.
   The keywords *if* , *elif* , and *else* are used for conditional statements.
   The keywords *is, in* used for checking condition with lists and booleans

   #1
   x = 2
   print(x ____ 2) # Return True
   print(x ____ 3) # Return False
   print(x ____ 3) # Return True

   #2
   name = "John"
   age = “23”
   ### Complete the condition ###
   < condition >
      print("Your name is John, and you are also 23 years old.")

   ### Complete the condition ###

```
< condition >
   print("Your name is either John or Rick.")
```

Please refer to gist sent separately about the exercise we did

3. Lists

   This object allows to define 1 or more objects under one variable and refer them as a whole or individuals.

   For example:

   ```
   my_list = [“aa”, 32, True, another_list, [‘aaa’]]
   ```

   Each location in the list is called index.
   Indexes start from “0” in python.

   Further read: https://www.w3schools.com/python/python_lists.asp

4. Tuples

   Tuple is a data structure very similar to the list.

   The main difference being that tuple manipulation are faster than list because tuples are immutable, which means once defined you cannot delete, add or edit any values inside it.

   Usage examples:

   Cases our data is not going to change- e.g week days.

   Pass the data to other object and protect it from mutation.

   Further read: https://www.w3schools.com/python/python_tuples.asp

5. Dictionary

   A dictionary works with keys and values instead of indexes.

   Main difference is that lists are single dimension while dict is two dimensional.

   Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it.

   dict = {“name”: “Foo”, age: 28, hobbies: [“bar”], “kids”: None}

   Further read: https://www.w3schools.com/python/python_dictionaries.asp

6. Looping in python

   There are two types of loops in python. The for loop and the while loop.

   - **for** loops are used when we want to repeat the same piece of code a fixed number of times.
   - **while** loops repeat as long as a certain boolean condition is met.

   In order to iterate with sequence of numbers we use the **range(n)**, function which is used to generate an array of number than we can iterate with for.

```
>>> range(5)
[0, 1, 2, 3, 4]

>>> range(2,6)
[2, 3, 4, 5]

>>> range(5, -1, -1)
[5, 4, 3, 2, 1, 0]

# example
for num in range(5):
        print(num)
```

```
# example
primes = [2, 3, 5, 7]
for prime in primes:
        print(prime)

# example
for x in range(3, 8, 2):
        print(x)
```

Further read: https://realpython.com/python-for-loop/

7. while loop:

```
count = 0
while count < 5:
        print(count)
        count += 1  # This is the same as count = count + 1
```

Further read: https://realpython.com/python-while-loop/

8. else after a loop

When the loop condition of "for" or "while" statement fails then code part in "else" is executed.
If break statement is executed inside a *for* loop then the "else" part is skipped.

Note that "else" part is executed even if there is a continue statement.

```
# example
count=0
while(count<5):
   print(count)
   count +=1
else:
   print("count value reached %d" %(count))
```

9. Break, Pass, Continue
   break is used to exit a for loop or a while loop;
   whereas **continue** is used to skip the current block, and return to the "for" or "while" statement.
   The **pass** statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet

Code from class:

```
count=0
while(count<5):
   print(count)
   count +=1
else:
   print("count value reached
{count}".format(count=count))


for i in range(1, 10):
   if(i%5==0):
      break
   print(i)
else:
   print("this is not printed because for loop is
terminated because of break but not due to fail
in condition")
```

10. Input
    For example if my program needs to know your name in order to greet you with a proper blessing, it can prompt you for input and than use it to show the full blessing:

    ```
    # example
    print("Enter your name:", end =" ")
    name = input()
    print("Have a good day " + name)
    ```

11. Modules

A module is a file containing Python definitions and statements.
The file name is the module name with the suffix .py appended.
In the module we can define different python definitions such as classes, functions, variables..

example for a module (several python files)

fibo.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Main.py

```
# Main file to use the module
import fibo
fibo.fib(1000)
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

12. Functions (methods)

Functions (also called methods) are a convenient way to divide your code into useful blocks.

Functions make the code more readable, and allows to reuse it.

Functions are a key way to define interfaces so programmers can share their code.

Functions can be “called” in different ways regarding their input and output.
Function can be defined in different variations:

- without any parameter
-get a simple parameter
- get as an input a complex variable such as dictionary or even another function, also any parameter can be defined with a default value.

When function finishes to execute it returns some sort of data.

It can be simple data like: booleans, strings, etc
It can be complex: a dictionary, an array or another function.

Will the following code work?

```
def no_return():
   x=5
   print(x)
```

Yes, because methods can return “None”, like in this case.

example:

```
dog_prefix = "Dog bark sounds like: "

def bark():
   print(dog_prefix + "Whoof! Whoof!")

def miao(catsound="Miao! Miao!"):
  return print(catsound)
```

```
def makeasound(sound):
  sound()

miao("Hatoola")
makeasound(bark)
makeasound(miao)
makeasound(partial(miao, "Miaoooooo!"))
```

Exercise from class:

1. Write a python program to print the square of all numbers from 0 to 10

2. Write a python program to find the sum of all even numbers from 0 to 10

3. Write a python program to read three numbers (a,b,c) and check how many numbers between ‘a’ and ‘b’ are divisible by ‘c’
4. Write a python program to get the following output

   1-----99

   2-----98

   3-----97

   . .

   . .

   98-----2

   99-----1

5. Write a python program to read four numbers (representing the four octets of an IP) and print the next five IP address

Eg:

Input:

192.168.255.252
----------Output---------
192.168.255.253
192.168 255.254
192.168 255.255
192.169.0.0
192.169.0.1

6. Write a python program to print the factorial of a given number

7. Write a python program to print the first 10 numbers Fibonacci series

8. Write a python program to read a number and print a right triangle using "*"

Eg :
Input : 5
----------Output---------
*
* *
* * *
* * * *
* * * * *

9. Write a python program to check given number is prime or not

10. Write a python program to print all prime numbers between 0 to 100 , and print how many prime numbers are there.

(you can find the solutions in the gist, or solve it by yourselves :))

---

13. Packages

A package is a pre-defined .py file (reminder- we discussed about modules)

- Contains variables and methods that we can **import** into our code and use it without thinking about it's implementation.
- A good example for that a package called ***datetime*** that we can use in order to execute date and time operations.

The difference between using only Import or using import and from is simple:

- Import only – let us use a module (file) with no specific class (as we learned a module can contain a few classes), in which case we will need to specify which class we want to use each time.

  >>> import time

```
>>> time.time()
```

- from and Import- will specify the module and class we want to use.

```
>>> from time import time
>>> time()
```

14. Using pip

Now we can easily install predefined packages from the internet and use them with our development process. Few examples:

```
pip install requests # install a new package called requests.

pip install --upgrade pandas

pip install --upgrade ansible
```