Python for Beginners

Glimpse of last lecture

Introduction (an overview to course)

- ~~Getting started "Hello World"~~
- ~~Keywords and identifiers~~
- ~~Comments and Statements~~
- ~~Variables and assignments~~
- ~~Data types~~
- Flow control
- Methods and Functions
- Reading and Writing files
- Modules and import
- Object Oriented Programming

Data types

- ~~Numbers~~
- ~~List~~
- ~~Tuple~~
- ~~String~~
- ~~Dictionary~~
- ~~Set~~

Program Flow Control in Python

- ~~*If* statement~~
- ~~Elif statement~~
- ~~More on if, elif and else~~
- *For* loop
- *While* loop
- ~~Useful operators~~

Methods and Functions

- Defining a function
- Flow when calling a function
- Parameters and arguments
- Global/local
- Functions calling functions

Reading/Writing Files

- Files and directories in Python
- Reading from a file
- Parsing data
- Printing data to external file

Modules and Import

- Standard Python library
- Datetime module
- Math and Random module
- Generators and decorators
- NumPy, Pandas, Matplotlib (basic uses)

Object Oriented Programming

- Introduction to OOPs
- Attributes and Class keywords
- Inheritance and Polymorphism

Statistical analysis of data with Python

Range in for loops

As per documentation, ‘ Range’ represents an immutable sequence of numbers and is commonly used for looping a specific number of times

Python ranges can be used in multiple forms

ranges(final) for x in range(11):

ranges(initial,final) for x in range(4,11):

ranges(initial,final,step)

- for x in range(4,11,1):
- for x in range(4,11,2):
- for x in range(11,1,-2):

Steps up / down

Nested FOR

Patters using FOR loop

Pattern 1

Nested FOR

Patters using FOR loop

Pattern 1


```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

Nested FOR

Patters using FOR loop

Pattern 1

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

Pattern 2

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

```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

Nested FOR

Patters using FOR loop

Pattern 1

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

Pattern 2

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

```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

```
for col in range(0, row+1):
```

Nested FOR

Patters using FOR loop

Pattern 1

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

Pattern 2

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

Pattern 3

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

```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

```
for col in range(0, row+1):
```

Nested FOR

Patters using FOR loop

Pattern 1

Pattern 2

Pattern 3


```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

```
for col in range(0, row+1):
```

```
for col in range(n-row,0,-1):
```

Nested FOR

Patters using FOR loop

Pattern 1

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

Pattern 2

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

Pattern 3

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

Pattern 4

```
1
1 2
1 2 3
1 2 3 4
```

```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

```
for col in range(0, row+1):
```

```
for col in range(n-row,0,-1):
```

Nested FOR

Patters using FOR loop

Pattern 1

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

Pattern 2

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

Pattern 3

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

Pattern 4

```
1
1 2
1 2 3
1 2 3 4
```

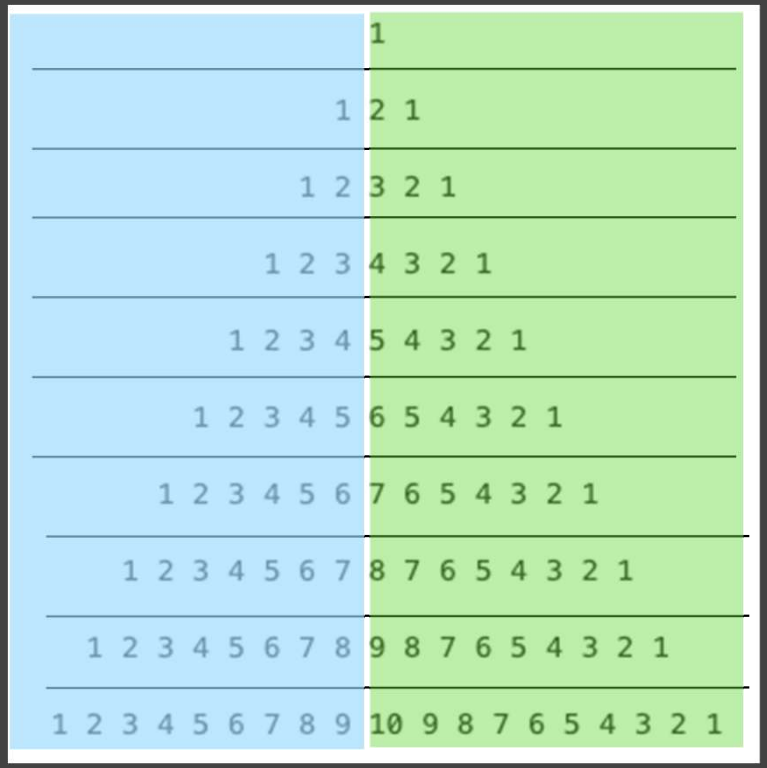
```
n = 5
for row in range(0, n):
  for col in range(0, n):
    print("*", end="")
  print()
```

```
for col in range(0, row+1):
```

```
for col in range(n-row,0,-1):
```

```
n = 5
for row in range(0, n):
  for col in range(1,row+1):
    print(col, end="")
  print()
```

Nested loop (Hands-On)



Nested loop (Hands-On)

```
for loop in range(1,11):
    for space in range(10-loop):
        print(" ", end = " ")
    for numbers in range(1,loop):
        print(numbers,end=" ")
    for numbers in range(loop,0,-1):
        print(numbers,end=" ")
    print("\n")
```

```
1                   1
2                 1 2 1
3               1 2 3 2 1
4             1 2 3 4 3 2 1
5           1 2 3 4 5 4 3 2 1
6         1 2 3 4 5 6 5 4 3 2 1
7       1 2 3 4 5 6 7 6 5 4 3 2 1
8     1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
9   1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
10 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1
```

While loop

for loop mainly use to iterate the actions through an iterable in a sequence or range .

but sometimes, one needs to iterate actions as long as certain condition is true.

As long as the condition remains true , the sequence of actions inside the loop will be executed

Once the condition becomes false, the loop terminates.

```
while <condition>:                    // Condition can evaluate to True or False
        block_of_code_to_execute
```

Print numbers between
1 – 10 using while loop

Print numbers between
1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 10-1 using while loop

Print numbers between 1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 10-1 using while loop

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Print numbers between 1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 10-1 using while loop

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Print numbers in the following format (nested while)

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Print numbers between 1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 10-1 using while loop

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Print numbers in the following format (nested while)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
row = 1
while row <= 5:

    num = 1
    while num <= row:
        print(num, end=" ")
        num = num + 1

    print()
    row = row + 1
```

Print numbers between 1 – 10 using while loop

Start

Condition

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Print numbers between 10-1 using while loop

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Print numbers in the following format (nested while)

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
row = 1
while row <= 5:

    num = 1
    while num <= row:
        print(num, end=" ")
        num = num + 1

    print()
    row = row + 1
```

```
    1
  1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

?

Print numbers between 1 – 10 using while loop

Start

```
num = 1
while num <= 10:
    print(num)
    num = num + 1
```

Condition

Print numbers between 10-1 using while loop

```
num = 10
while num >= 1:
    print(num)
    num = num - 1
```

Print numbers in the following format (nested while)

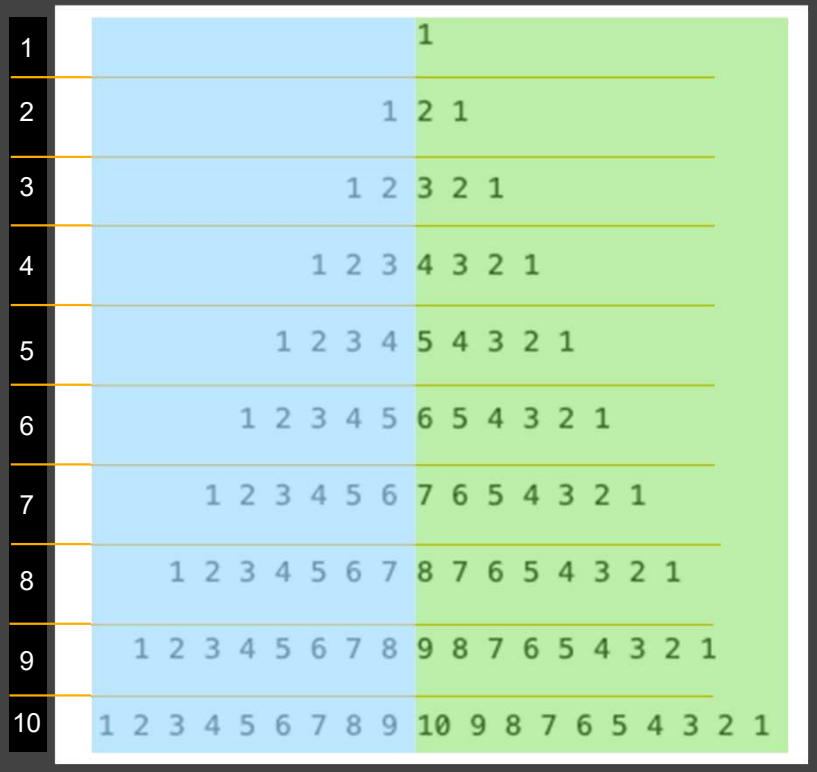
```
    1
  1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
rows = 5
row = 1
while row <= rows:
    space = 1
    while space <= rows - row:
        print("  ", end="") # two spaces
        space = space + 1

    num = 1
    while num <= row:
        print(num, end=" ")
        num = num + 1

    print()
    row = row + 1
```

- Reproduce using while loop -



- Reproduce using while loop -

```
x = 1
space = 1
LeftHalf = 1
while x < 11:
    while space < (11-x):
        print(" ",end=" ")
        space=space+1
    while LeftHalf < x:
        print(LeftHalf,end=" ")
        LeftHalf=LeftHalf+1
    RightHalf = x
    while RightHalf > 0:
        print(RightHalf,end=" ")
        RightHalf-=1
    x=x+1
    print("\n")
    space=1
    LeftHalf=1
```

```
                    1
                  1 2 1
                1 2 3 2 1
              1 2 3 4 3 2 1
            1 2 3 4 5 4 3 2 1
          1 2 3 4 5 6 5 4 3 2 1
        1 2 3 4 5 6 7 6 5 4 3 2 1
      1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
    1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
  1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1
```

One can conclude, *while* loops are more flexible since one explicitly set the start and end conditions, but require *more set-up* than *for* loops

- More on while loop -

while loop is frequently used when one cannot determine/know in advance how often the loop must be executed, however in ***for*** loop, each item repeats in pre-determined range/sequence.

*The **while loop** is especially important and is used when*

- ✓ *Reading data from files where you do not know in advance how much data to process.*
- ✓ *The **while loop** continue reading until there is no more data, or it terminates when the desired condition is met.*

-Uses of while loop -

Hands-On

Use a `while` loop to generate a random number between 1 and 14 until you get the number 7, print some message.

- First condition, how many times you want to check
- Generate random number and save it to a variable
- Check if variable value is equal to 7, print message for both cases (i) when generate number is 7 or
  (ii) generate number is not 7

#Hint:

#How to generate random number from random

```
import randint
randint(a, b)        # Generate random number between a, b
```

```
from random import randint
i = 1
while i < 30:
    number = randint(1,14)
    if number == 7:
        print("found the number")
        break
    elif number != 7:
        print("Please try again")
    i+=1
```

- Functions -

Similar to loops, functions also allow to execute the **S**ame **T**ask **S**everal **T**imes

Don't **R**epeat **Y**ourself

- Functions -

Similar to loops, functions also allow to execute the Same Task Several Times

Don't Repeat Yourself

A function is a *block of statements* that performs a particular task.

- Functions provide **better modularity** and a **high degree of code reusing** (avoid duplicating).
- Functions can be 'abstracted' for the other users

  e.g. print()

- Functions -

Similar to loops, functions also allow to execute the **Same Task Several Times**

Don't Repeat Yourself

A function is a *block of statements* that performs a particular task.

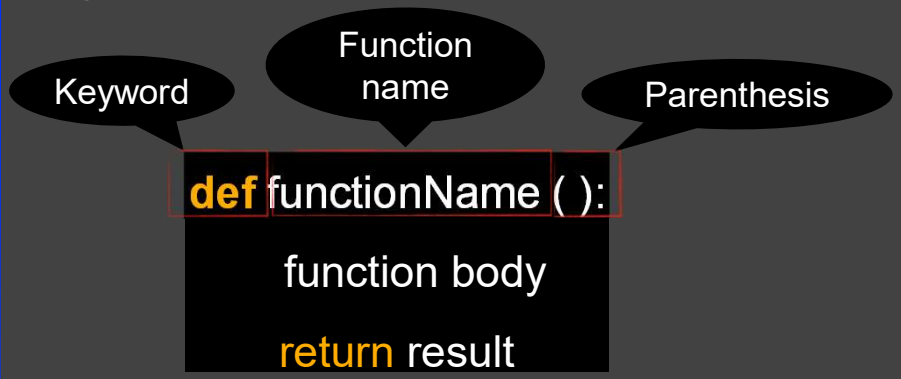
- Functions provide better modularity and a high degree of code reusing (avoid duplicating).
- Functions can be 'abstracted' for the other users

  e.g. print()

  - Define and use the function - (difference between function - method (OOP))
  - Interacting with functions (input / output / return)

- Define the function -

Syntax



- Define the function -

Syntax

Keyword

Function name

Parenthesis

```
def functionName ( ):
    function body
    return result
```

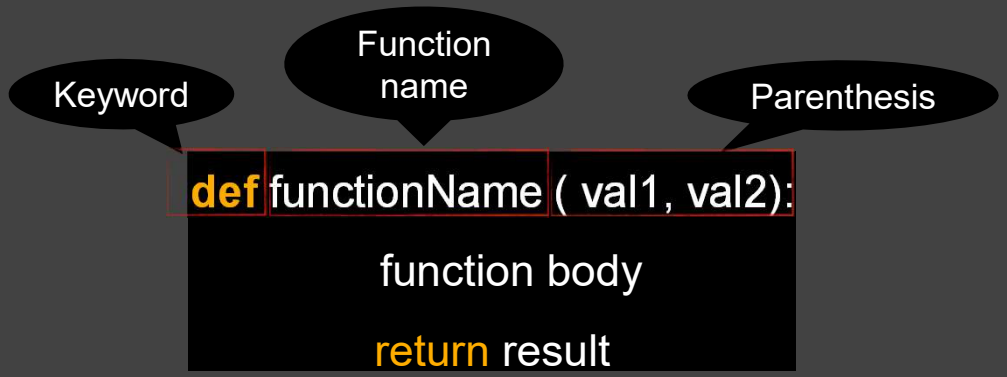
```
def sum( ):
    result = 10 + 25
    print("Value which will return is = ",result)
    return result

sum( )
```

- Passing parameters / arguments -

Parameters : work as *placeholders* for the values *pass to the function*. So, they are like variables which are used to pass the value when one call the function.

Syntax



- Using the functions -

```
def sum(val1,val2 ):
    result = val1 + val2
    print("Value which will return is = ",result)
    return result
sum( 20, 15)
```

```
def sum(val1, val2, val3, val4, val5 ):
    add = val1 + val2
    mul  = val3 * val4 * val4
    result = add + mul
    print("Value which will return is = ",result)
    return result
sum( 2,  3, 7, 10.5, 2.5)
```

Using function, print the sum of even numbers between 1 - 100

```
for num in range(1,100):
    result = Num_type(num)
    SumOfEvenNumbers = SumOfEvenNumbers + result
print("Sum of even numbers = ",SumOfEvenNumbers)
```

Using function, print the sum of even numbers between 1 - 100

```
SumOfEvenNumbers=0
null=0
def Num_type(val):
    check_num = val % 2
    if check_num == 0:
        return val
    else:
        return null

for num in range(1,100):
    result = Num_type(num)
    SumOfEvenNumbers = SumOfEvenNumbers + result
print("Sum of even numbers = ",SumOfEvenNumbers)
```

Functions - to be continued