Tutorial_Session1

March 17, 2018

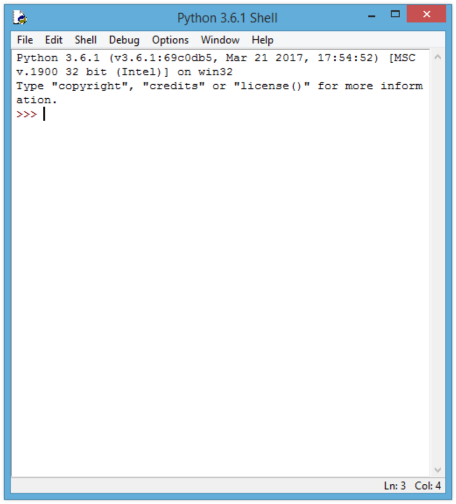
---

1 PYTHON

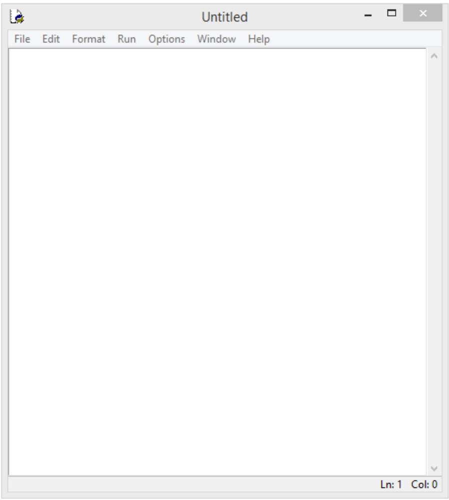
- Interactive, interpreted, and object-oriented programming language.
- Simple syntax
- Developed by Guido Van Rossum in 1991 at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Name was inspired by: Monty Python's Flying Circus

1.1 PYTHON PROGRAMMING ENVIRONMENT

- Available on a wide variety of platforms including Windows, Linux and Mac OS X.
- Official Website: **python.org**
- IDLE stands for Integrated Development and Learning Environment. Python IDLE comprises Python Shell and Python Editor. Python Shell Python Editor



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> 
```



1.2 Display on screen

```
In [2]: print('hello world')
```

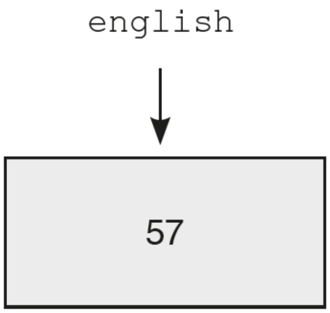
```
hello world
```

1.3 Names (Variables) and Assignment Statements

- Variables provide a means to name values so that they can be used and manipulated later.
- Assignment Statement: Statement that assigns value to a variable.

```
In [ ]: english = 57
        print(english)
```

Python associates the **name** (variable) **english** with value **57** i.e. the name (variable) **english** is assigned the value **57**, or that the name (variable) **english** refers to value **57**. Values are also called **objects**.



1.3.1 Rules for creating a name (variable)

- Must begin with a letter or _ (underscore character)
- May contain any number of letters, digits, or underscore characters. No other character apart from these is allowed.

1.3.2 Shorthand Notation

```
        a = a <operator> b              is equivalent to
        a <operator>= b
```

```
In [3]: a = 6
        a = a + 5
        print(a)
        a = 6
        a += 5
        print(a)
```

```
11
11
```

1.3.3 Multiple Assignments

- Used to enhance the readability of the program.

```
In [7]: msg, day, time = 'Meeting', 'Mon', '9'
        totalMarks = count = 0
```

1.4 Arithmetic Operators

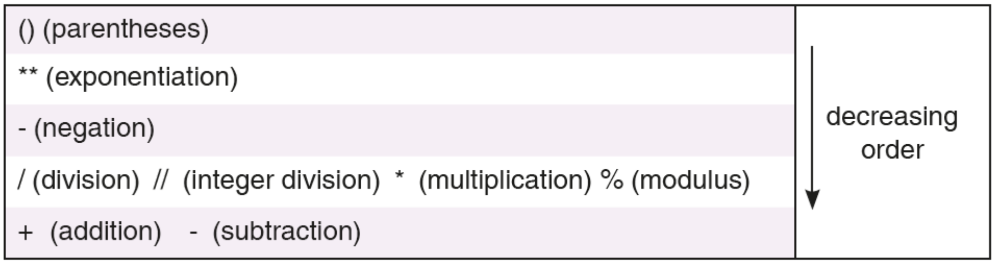
```
In [11]: print("18 + 5 =", 18 + 5)     #Addition
         print("18 - 5 =", 18 - 5)     #Subtraction
         print("18 * 5 =", 18 * 5)     #Multiplication
         print("27 / 5 =", 27 / 5)     #Division
         print("27 // 5 =", 27 // 5)   #Integer Division
         print("27 % 5 =", 27 % 5)     #Modulus
         print("2 ** 3 =", 2 ** 3)     #Exponentiation
         print("-2 ** 3 =", -2 ** 3)   #Exponentiation
```

```
18 + 5 = 23
18 - 5 = 13
18 * 5 = 90
27 / 5 = 5.4
27 // 5 = 5
27 % 5 = 2
2 ** 3 = 8
-2 ** 3 = -8
```

```
In [9]: print("'how' + ' are' + ' you?':", 'how' + ' are' + ' you?')
        print("'hello' * 5             :", 'hello' * 5)
```

```
'how' + ' are' + ' you?': how are you?
'hello' * 5             : hellohellohellohellohello
```

1.4.1 Precedence of Arithmetic Operators



() (parentheses)	↓ decreasing order
** (exponentiation)	
- (negation)	
/ (division) // (integer division) * (multiplication) % (modulus)	
+ (addition) - (subtraction)	

1.5 Relational Operators

- Used for comparing two expressions and yield True or False.
- The arithmetic operators have higher precedence than the relational operators.

```
In [ ]: print("23 < 25  :", 23 < 25)           #less than
        print("23 > 25  :", 23 > 25)           #greater than
        print("23 <= 23 :", 23 <= 23)          #less than or equal to
        print("23 - 2.5 >= 5 * 4 :", 23 - 2.5 >= 5 * 4) #greater than or equal to
        print("23 == 25 :", 23 == 25)          #equal to
        print("23 != 25 :", 23 != 25)          #not equal to
```

- When the relational operators are applied to strings, strings are compared left to right, character by character, based on their ASCII codes, also called ASCII values.

```
In [12]: print("'hello' < 'Hello' :", 'hello' < 'Hello')
         print("'hi' > 'hello'     :", 'hi' > 'hello')
```

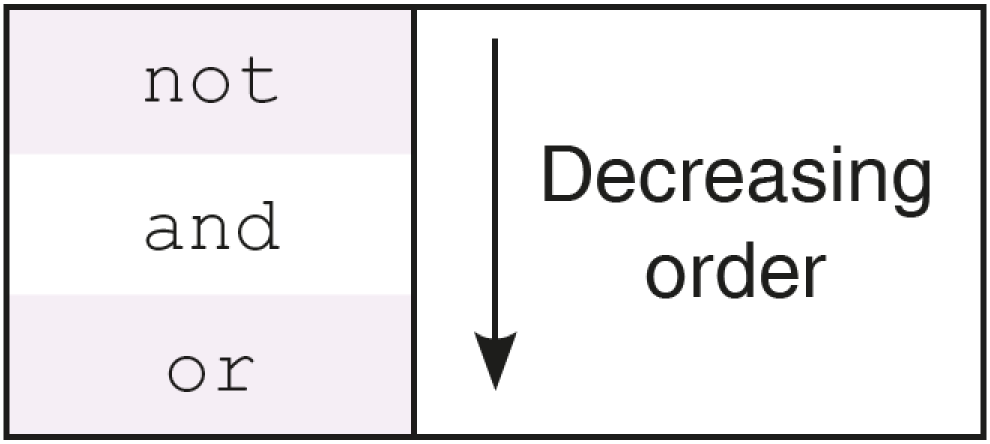
```
'hello' < 'Hello' : False
'hi' > 'hello'     : True
```

1.6 Logical Operators

- The logical operators not, and, and or are applied to logical operands True and False, also called Boolean values, and yield either True or False.
- As compared to relational and arithmetic operators, logical operators have the least precedence level.

```
In [ ]: print("not True < 25       :", not True)              #not operator
        print("10 < 25 and 5 > 6  :", 10 < 25 and 5 > 6) #and operator
        print("10 < 25 or 5 > 6   :", 10 < 25 or 5 > 6)   #or operator
```

1.6.1 Precedence of Logical Operators



1.7 Python Keywords

- Reserved words that are already defined by the Python for specific uses.

```
In [ ]: import keyword
        print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class','continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda','nonlocal', 'not', 'or', 'pass',
'raise', 'return', 'try', 'while', 'with', 'yield']
```

1.8 Functions

- Functions provide a systematic way of problem solving by dividing the given problem into several sub-problems, finding their individual solutions, and integrating the solutions of individual problems to solve the original problem.
- This approach to problem solving is called stepwise refinement method or modular approach.

1.9 Built-in Functions

- Predefined functions that are already available in Python.

1.9.1 Type Conversion: int, float, str functions

```
In [13]: str(123)

Out[13]: '123'

In [14]: int('234')

Out[14]: 234

In [15]: int(234.8)

Out[15]: 234
```

1.9.2 input function

- Enables us to accept an input string from the user without evaluating its value.
- The function input continues to read input text from the user until it encounters a newline.

```
In [16]: name = input('Enter a name: ')
         print('Welcome', name)
```

```
Enter a name: Suresh
Welcome Suresh
```

```
In [21]: costPrice = int(input('Enter cost price: '))
         profit = int(input('Enter profit: '))
         sellingPrice = costPrice + profit
         print('Selling Price: ', sellingPrice)
```

```
Enter cost price: 50
Enter profit: 12
Selling Price:  62
```

1.9.3 eval function

- Used to evaluate the value of a string.

```
In [22]: a = eval('15+10')
         print(a)
```

```
25
```

1.9.4 min and max functions

- Used to find maximum and minimum value respectively out of several values.

```
In [23]: a = max(59, 80, 95.6, 95.2)
         b = min('hello', 'how', 'are', 'you', 'Sir')
         print("Minimum Value", a)
         print("Maximum Value", b)
```

```
Minimum Value 95.6
Maximum Value Sir
```

1.9.5 Functions from math module

- Used to find maximum and minimum value respectively out of several values.

```
In [25]: import math
         print("math.ceil(3.4)  :", math.ceil(3.4))
         print("math.pow(3, 3)  :", math.pow(3, 3))
         print("math.sqrt(65)   :", math.sqrt(65))
         print("math.sqrt(65)   :", round(math.sqrt(65),2))
         print("math.log10(100) :", math.log10(100))
```

```
math.ceil(3.4)  : 4
math.pow(3, 3)  : 27.0
math.sqrt(65)   : 8.06225774829855
math.sqrt(65)   : 8.06
math.log10(100) : 2.0
```

1.9.6 help function

- Used to know the purpose of a function and how it is used.

```
In [26]: import math
         print(help(math.cos))
```

```
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).

None
```

1.10 Function Definition and Call

The **syntax** for a function definition is as follows:

```
def function_name ( comma_separated_list_of_parameters):
    statements
```

Note: Statements below **def** begin with four spaces. This is called **indentation**. It is a requirement of Python that the code following a colon must be indented.

```
In [28]: def triangle():
             '''
             Objective: To print a right angled triangle.
             Input Parameter: None
             Return Value: None
             '''
             '''
             Approach: To use a print statement for each line of output
             '''
             print('*')
             print('* *')
             print('* * *')
             print('* * * *')
```

Invoking the function

```
In [29]: triangle()
```

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

1.10.1 Computing Area of the Rectangle

```
In [30]: def areaRectangle(length, breadth):
             '''
             Objective: To compute the area of rectangle
             Input Parameters: length, breadth  numeric value
             Return Value: area - numeric value
             '''
             area = length * breadth
             return area
```

```
In [33]: areaRectangle(7,5)
```

```
Out[33]: 35
```

```
In [34]: help(areaRectangle)
```

```
Help on function areaRectangle in module __main__:

areaRectangle(length, breadth)
    Objective: To compute the area of rectangle
    Input Parameters: length, breadth  numeric value
    Return Value: area - numeric value
```

```
In [43]: def areaRectangle(length, breadth=1):
             '''
             Objective: To compute the area of rectangle
             Input Parameters: length, breadth - numeric value
             Return Value: area - numeric value
             '''
             area = length * breadth
             return area

         def main():
             '''
             Objective: To compute the area of rectangle based on user input
             Input Parameter: None
             Return Value: None
             '''
             print('Enter the following values for rectangle:')
             lengthRect = int(input('Length : integer value: '))
             breadthRect = int(input('Breadth : integer value: '))
             areaRect = areaRectangle(lengthRect, breadthRect)
             print('Area of rectangle is', areaRect)

         if __name__ == '__main__':
             main()
```

```
Enter the following values for rectangle:
Length : integer value: 7
Breadth : integer value: 5
Area of rectangle is 35
```

1.11 Control Structures

- Needed for non-sequential and repetitive execution of instructions.

1.12 if Conditional Statement

- Used to execute a certain sequence of statements depending upon fulfilment of a particular condition > The general form of **if-elif-else** statement is as follows:

  if < condition1 >: < Sequence S1 of statements to be executed > elif < condition2 >: < Sequence S2 of statements to be executed > elif < condition3 >: < Sequence S3 of statements to be executed > ...

  else: < Sequence Sn of statements to be executed >

1.12.1 Problem: Grade assignment on the basis of marks obtained

```
In [44]: def assignGrade(marks):
             '''
             Objective: To assign grade on the basis of marks obtained
             Input Parameter: marks  numeric value
             Return Value: grade - string
             '''
             assert marks >= 0 and marks <= 100
             if marks >= 90:
                 grade = 'A'
             elif marks >= 70:
                 grade = 'B'
             elif marks >= 50:
                 grade = 'C'
             elif marks >= 40:
                 grade = 'D'
             else:
                 grade = 'F'
             return grade

         def main():
             '''
             Objective: To assign grade on the basis of input marks
             Input Parameter: None
             Return Value: None
             '''
```

```
            marks = float(input('Enter your marks: '))
            print('Marks:', marks, '\nGrade:', assignGrade(marks))

        if __name__ == '__main__':
            main()
```

```
Enter your marks: 89
Marks: 89.0
Grade: B
```

1.13 for Statement

- It is used when we want to execute a sequence of statements (indented to the right of keyword for) a fixed number of times. > Syntax of **for** statement is as follows:

  for variable in sequence:

```
In [38]: for letter in "hello":
             print(letter)
```

```
h
e
l
l
o
```

1.13.1 Generating sequence of numbers using range function

Syntax:

```
range(start, end, increment)
```

The function call **range(1,n + 1)** produces a sequence of numbers from 1 to n

```
In [ ]: start = 1
        limit = 11
        for num in range(start, limit):
            print(num)
```

```
In [ ]: start = 1
        limit = 11
        step = 2
        for num in range(start, limit, step):
            print(num)
```

```
In [ ]: start = 30
        limit = -4
        step = -3
        for num in range(start, limit, step):
            print(num)
```

```
In [ ]: limit = 5
        for num in range(limit):
            print(num)
```

1.13.2 Problem: Printing a Triangle

```
In [1]: def rightTriangle(rows):
            '''
            Objective: To print a triangle comprising of asterisks
            Input Parameter: rows - numeric
            Return Value: None
            '''
            for i in range(1, rows + 1):
                print('*' * i)

        def main():
            '''
            Objective: To compute factorial of a number provided as an input
            Input Parameter: None
            Return Value: None
            '''
            rows = int(input('Enter number of rows: '))
            rightTriangle(rows)

        if __name__ == '__main__':
            main()
```

```
Enter number of rows: 6
*
**
***
****
*****
******
```

1.13.3 Problem: Factorial of a number

```
In [41]: def factorial(num):
             '''
             Objective: To compute factorial of a number
             Input Parameter: num - numeric
             Return Value: num! - numeric
             '''
             if num <= 0:
                 return 'Factorial Not defined'
             fact = 1
             for i in range(1, num+1):
```

```
            fact = fact * i
        return fact

    def main():
        '''
        Objective: To compute factorial of a number provided as an input
        Input Parameter: None
        Return Value: None
        '''
        num = int(input('Enter the number: '))
        fact = factorial(num)
        print("Result:", fact)

    if __name__ == '__main__':
        main()
```

```
Enter the number: 5
Result: 120
```

1.14 while Statement

- It is used for executing a sequence of statements again and again on the basis of some test condition.
- If the test condition holds True, the body of the loop is executed, otherwise the control moves to the statement immediately following the while loop. > Syntax of **while** statement is as follows:

  while :

```
In [ ]: count, n = 1, 5
        while count < n+1:
            print(count)
            count += 1
```

1.14.1 Sum of digits of a number

```
In [45]: def sumOfDigits(num):
             '''
             Objective: To compute sum of digits of a number
             Input Parameter: num - numeric
             Return Value: numeric
             '''
             '''
             Approach:
                 Ingore the sign of number. Initialize sum to zero.
                 Extract digits one by one beginning unit's place and keeps on
                 adding it to sum.
             '''
```

```
    num = abs(num)
    total = 0
    while num >= 1:
        total += (num % 10)
        num = num // 10
    return total

def main():
    '''
    Objective: To compute sum of digits of a number provided as an input
    Input Parameter: None
    Return Value: None
    '''
    num = int(input('Enter the number: '))
    total = sumOfDigits(num)
    print("Result:", total)

if __name__ == '__main__':
    main()
```

```
Enter the number: 123
Result: 6
```