PYTHON LECTURES

Python Programming Language
Assignment Statement
Input & Output

If Statement    Looping    While Statement
For Statement 2016–2017

http://www.aiet.edu.eg/StudentArea.asp
the aim of this topic is helping you to be a programmer – perhaps not a professional programmer, but at least you will have the skills to look at a data/information analysis problem and develop a program to solve the problem.

In a sense, you need two skills to be a programmer:
- First, you need to know the programming language (Python) – you need to know the vocabulary and the grammar.
- You need to be able to spell the words in this new language properly and know how to construct well-formed “sentences” in this new language.
- Second, you need to “tell a story”. In writing a story, you combine words and sentences to convey an idea to the reader. There is a skill and art in constructing the story, and skill in story writing is improved by doing some writing and getting some feedback. In programming, our program is the “story” and the problem you are trying to solve is the “idea”.

Introduction to python
The >>>> prompt is the Python interpreter’s way of asking you, “What do you want me to do next?” Python is ready to have a conversation with you.
Programming basics

- **code or source code**: The sequence of instructions in a program.

- **syntax**: The set of legal structures and commands that can be used in a particular programming language.

- **output**: The messages printed to the user by a program.

- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.
Topics:

- Constants: Numbers, Strings
- Variables
- Lists
- Operations and Expressions
- Assignment Statement
- Input Statements
- Print Statement
BASICS

Values
- Integer: 42
- Float (real number): 12.345
- String (of characters): "Hello!"

Types
- Integer
- Float (real number)
- String (of characters)

Expressions
- Integer: 34 * (23 + 14)
- Float: 1.0 / 3.0
- String: "Hello" + "lo!"
Once you’ve saved the program, it will run. The program will ask you for a temperature. Type in 20 and press enter. The program’s output looks something like this:, . The first line asks the user to enter a temperature. The input function’s job is to ask the user to type something in and to capture what the user types.

Enter a temperature in Celsius: 20
In Fahrenheit, that is 68.0

A second program

The eval function is something we use it when we’re getting numerical input. The eval function converts the text entered by the user into a number. One nice feature of this is you can enter expressions, like 3*12+5, and eval will compute them for you.

Printing example

The value of 3+4 is 7
A 1 XYZ 2
Python Syntax

Python is an easy to learn, powerful high level programming language.

- Comments: `# This is a comment`
- Constants:
  - Numeric (عددی)
    - Integers like: `1`, `-1`, `9`, `2334`, `-90`
    - Real numbers like: `90.89`, `-23.9`, `89.00`
    - Exponential notation like: `90E10` (i.e. `90 * 10^{10}`), `0.827E-27` (i.e. `0.827 * 10^{-27}`)
  - Strings (حرفی)
    - A string is a sequence of characters
    - "Hello World"
    - `Enter the number please`
Variables in Python

- Variable is a storage location associated with symbolic name which contains value
- A variable name is a sequence of characters which should start with a letter, followed by any combination of letters and digits, **Cannot be one of the reserved Python words**
- Variable: Usage:
  - Compute an expression's result,
  - store that result into a variable,
  - and use that variable later in the program.

- A Variable should be given a value before using it.
- **Examples:**
  - `width = 20`
  - `x=y=z=0`  # the values of x, y, and z are 0
  - `a = 1; b=2`
Assignment: Var = Expr

- An expression is a **formula** consisting of many terms joined by operators.
- The **right hand-side** is **evaluated** first and the value is **assigned** to the variable in the **left hand-side**.
- **Examples**
  - X = Y + Z
  - Area = L * L
  - Y = (5 + math.sin(X)) / 2  # Built-in functions
  - Length = 2
  - X = "Mohamed Abbas"
Operators

Arithmetic Operators
+    Addition
-    Subtraction
*    Multiplication
/    Division
**   Power
%    Modulo

Relational Operators
==   Equal to
<>   ‘!=’ Not equal to
<    Less than
<=   Less than or equal to
>    Greater than
>=   Greater than or equal to

Logical Operators
and  And operator
or   Or operator
not  For negation

String Operator
+    Concatenation
*    Replication
Order of Evaluation

Follow that order:
1. Evaluate the parentheses first.
2. Exponentiation.
3. Multiplication, Division and Modulo.
4. Addition and Subtraction.

When two operations of the same order exist in an expression, the order of evaluation of them is from left to right.
Arithmetic expression evaluation

Example 1:
\[ 3 + (2 - 4) \times 5^2 \]
\[ = 3 + (-2) \times 5^2 \]
\[ = 3 + (-2) \times 25 \]
\[ = 3 - 50 \]
\[ = -47 \]

Example 2:
\[ 3 + 4 - 2 \times 5^2 \]
\[ = 3 + 4 - 2 \times 25 \]
\[ = 3 + 4 - 50 \]
\[ = 7 - 50 \]
\[ = -43 \]
Logical expressions have either True or False result

Examples:
- not (5==5 and 5<4)
- not (True and 5<4)
- not (True and False)
- not (False)
- True

not 5==5 and 5>4
not True and 5>4
False and 5>4
False and True
False

6*5==2*15 or (7-3)/2==2
6*5==2*15 or 4/2==2
30==2*15 or 4/2==2
30==30 or 4/2==2
30==30 or 2==2
True or 2==2
True or True
True
Examples of operators

Anything after a # symbol is treated as a comment
Floor division operator // (integer division)

Operators

- Example operators.py
  ```python
  print 2*2
  print 2**3
  print 10%3
  print 1.0/2.0
  print 1/2
  ```

  Output:
  4
  8
  1
  0.5
  0

- Note the difference between floating point division and integer division in the last two lines
Examples of Expressions, arithmetic operators, and precedence

```python
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6

>>> 17 / 3  # classic division returns a float
5.66666666666667

>>> 17 // 3  # floor division discards the fractional part
5

>>> 17 % 3  # the % operator returns the remainder of the division
2

>>> 5 * 3 + 2  # result * divisor + remainder
17
```
If a variable is not "defined" (assigned a value), trying to use it will give you an error:
Math commands

- Python has useful **commands** for performing calculations.

<table>
<thead>
<tr>
<th>Command name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>ceil(value)</td>
<td>rounds up</td>
</tr>
<tr>
<td>cos(value)</td>
<td>cosine, in radians</td>
</tr>
<tr>
<td>floor(value)</td>
<td>rounds down</td>
</tr>
<tr>
<td>log(value)</td>
<td>logarithm, base e</td>
</tr>
<tr>
<td>log10(value)</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>max(value1, value2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>min(value1, value2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>sin(value)</td>
<td>sine, in radians</td>
</tr>
<tr>
<td>sqrt(value)</td>
<td>square root</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>pi</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>

To use many of these commands, you must write the following at the top of your Python program:

```python
import math *
```
Key words (reserved words)

Python reserves 33 keywords:

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

You might want to keep this list handy. If the interpreter complains about one of your variable names and you don’t know why, see if it is on this list.
Conditionals control

- True and false Booleans
- Comparison and logical operators
- If, elif, and else statement
A Boolean expression can be evaluated as True or False.

An expression evaluates to False, if it is... the constant False, the object None, an empty sequence or collection, or a numerical item of value 0

Everything else is considered True
Logical operators

- AND, OR, NOT
- `>>=` 2+2==5 or 1+1==2  TRUE
- `>>=` 2+2==5 and 1+1==2  FALSE
- `>>=` not(2+2==5) and 1+1==2  TRUE
print

print : Produces text output on the console.

Syntax:

print "Message"
print Expression

- Prints the given text message or expression value on the console, and moves the cursor down to the next line.

print Item1, Item2, ..., ItemN

- Prints several messages and/or expressions on the same line.

Examples:

print "Hello, world!"
age = 45
print "You have", 65 - age, "years until retirement"

Output:

Hello, world!
You have 20 years until retirement
**input**

- **input**: Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - **Example**:
    ```python
    age = eval(input("How old are you? "))
    print "Your age is", age
    print "You have", 65 - age, "years until retirement"
    ```

  **Output**:

  ```plaintext
  How old are you? 53
  Your age is 53
  You have 12 years until retirement
  ```

- **Exercise**: Write a Python program that prompts the user for his/her amount of money, then reports how many Nintendo Wiis the person can afford, and how much more money he/she will need to afford an additional Wii.
Order of Evaluation

Follow that order:
1. Evaluate the parentheses first.
2. Exponentiation.
3. Multiplication, Division and Modulo.
4. Addition and Subtraction.

When two operations of the same order exist in an expression, the order of evaluation of them is from left to right.
Repetition (loops) and Selection (if/else)
Control flow Statements in Python

- If Statement
- While Statement
- For Statement
if Statement

- **Condition:**
  - condition may consist of one, or more expressions connected by a relational operator
  - It evaluates to **True** or **False**
    - Examples:
      - \( x == y \)
      - \((a > b) \text{ and } (c < d)\)

- **if statement:** Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
  - Syntax:
    - if condition:
      - statements

**Example:**
```python
x = 3.4
if x > 2.0:
    print("Your application is accepted.")
```
Indentation = Tab

Statement Block is any of these statements:

- Assignment Statement
- INPUT Statement
- PRINT Statement
- If Statement
- While Statement
- For Statement
Examples

If the color is blue
   I will buy the car
or else
   I will not buy the car

If the price is cheap
   I will buy the car
or else
   I will not buy the car

If the color is blue or the price is cheap
   I will buy the car
or else
   I will not buy the car

If the color is blue or the price is cheap
   I will buy the car
or else if the color is grey
   I will think about buying the car
or else
   I will not buy the car

Decisions in a programming language. How??
An if statement tells the computer to take one of two alternative courses of action, depending on whether the value of a given Boolean-valued expression is true or false.

```python
if <condition1>:
    <Indentation> [statementBlock−1]
elif <condition2>:
    <Indentation> [statementBlock−2]]...
else:
    <Indentation> [statement Block−n]...
```
If the color is blue or the price is cheap
I will buy the car
or else
I will not buy the car
Color = "blue"; price = 10000
If color == "blue" or price <= 15000:
    print ("I will buy the car")
else:
    print ("will buy the car")

Program’s Output: I will buy the car
Write a program that determines whether an integer number N is odd or even.

Solution:

```python
n = eval(input("enter a number"))
If n % 2 == 0:
    print "n" + "is even"
else:
    print "n" + "is odd"
```

Output

```python
>>> Enter a number 25
25 is odd
```
Read two numbers a, b and then print them in an ascending order.
(For example: if the inputs are 9, 5 → the output 5, 9)

Solution

```python
a = eval(input("enter number a: "))
b = eval(input("enter number b: "))
If a > b:
    c = a
    a = b
    b = c
Print("numbers in ascending order", a, " ", b)
```

Output

```python
>>>
enter number a: 25
enter number b: 14
numbers in ascending order" 14  25
```
The for loop

- **for loop**: Repeats a set of statements over a group of values.
- **Syntax**:
  ```python
define variable Name in group Of Values:
  statements
  ```
- We indent the statements to be repeated with tabs or spaces.
- Variable Name gives a name to each value, so you can refer to it in the statements.
- Group Of Values can be a range of integers, specified with the `range` function.
- **Example**:
  ```python
  for x in range(1, 6):
    print x, "squared is", x * x
  ```
  
  **Output**:
  1 squared is 1
  2 squared is 4
  3 squared is 9
  4 squared is 16
  5 squared is 25
range

The range function specifies a range of integers:

- \texttt{range}(\texttt{start}, \texttt{stop}) \quad \text{– the integers between}
- \texttt{start (inclusive)} \quad \text{and stop (exclusive)}

- It can also accept a third value specifying the change between values.
  - \texttt{range}(\texttt{start}, \texttt{stop}, \texttt{step}) \quad \text{– the integers between}
  - \texttt{start (inclusive)} \quad \text{and stop (exclusive) by step}

- Example:
  
  ```python
  for x in range(5, 0, -1):
    print x
  print "OK!"
  ```

Output:

5
4
3
2
1
OK!
Cumulative loops

Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a *cumulative sum*.

```python
sum = 0
for i in range(1, 11):
    sum = sum + (i * i)
print "sum of first 10 squares is", sum
```

Output:
```
sum of first 10 squares is 385
```

**Exercise:** Write a Python program that computes the factorial of an integer.
while

- **while** loop: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)

**Syntax:**

```python
while condition:
  statements
```

**Example:**

```python
number = 1
while number < 200:
  print number,
  number = number * 2
```

- **Output:**
  
  1 2 4 8 16 32 64 128
String properties

- `len(string)` - number of characters in a string (including spaces)
- `str.lower(string)` - lowercase version of a string
- `str.upper(string)` - uppercase version of a string

Example:
name = "Martin Douglas Stepp"
length = `len(name)`
big_name = `str.upper(name)`
print big_name, "has", length, "characters"

Output:
MARTIN DOUGLAS STEPP has 20 characters