Dr. Sivanthi Aditanar College of Engineering
Tiruchendur – 628205
Department of Information Technology

Organised By

One Week STTP on "Python Programming"
(22 June 2020 to 28 June 2020)

DAY-1 AND DAY-2 NOTES (22-06-2020 and 23-06-2020)

- Execute the programs in different modes.
    - Interactive Mode Programming
    - Script Mode Programming

- Once Python is installed, typing python in the command line will invoke the interpreter in immediate mode.
- We can directly type in Python code, and press Enter to get the output.
- This prompt can be used as a calculator.
- To exit this mode, type quit() and press enter.
- IDE - Integrated Development Environment
- When you install Python, an IDE named IDLE is also installed
- We can use any text editing software to write a Python script file.
- When you open IDLE, an interactive Python Shell is opened.
- Now you can create a new file and save it with .py extension.
- Write Python code in the file and save it.
- To run the file, go to Run > Run Module or simply click F5.

Keywords:

- Keywords are the reserved words in Python.
- We cannot use a keyword as a variable name, function name or any other identifier.
- In Python, keywords are case sensitive.
- Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore **_**.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- We cannot use special symbols in identifiers.
- An identifier can be of any length.
- Python is a case-sensitive language.
- Multiple words can be separated using an underscore.

Statement:

- Instructions that a Python interpreter can execute are called statements.

Multi-line statement

- The end of a statement is marked by a newline character (\).
  - **a = 1 + 2 + 3 + **
    - **4 + 5 + 6 + **
    - **7 + 8 + 9**
- Multiple statements in a single line using semicolons.
  a = 1; b = 2; c = 3
- Python does not use braces({}) to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- We use the hash (**#**) symbol to start writing a comment.
- All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.
- Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

Data Types:

- The data stored in memory can be of many types.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary
- Number data types store numeric values.
- Number objects are created when you assign a value to them.
- You can also delete the reference to a number object by using the **del** statement.
- You can delete a single object or multiple objects by using the **del** statement.
- Python supports three different numerical types –
  - int (signed integers)
  - float (floating point real values)
  - complex (complex numbers)
- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows either pair of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:])
- Indexes starting at 0 in the beginning of the string and working their way from -1 to the end.

- The plus **(+)** sign is the string concatenation operator.
- The asterisk **(*)** is the repetition operator.
- Lists are the most versatile of Python's compound data types.
- List contains items separated by commas and enclosed within square brackets ([]).
- Lists are similar to arrays in C.
- One of the differences between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:])
- Indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus **(+)** sign is the list concatenation operator.
- The asterisk **(*)** is the repetition operator.
- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas.
- Unlike lists, however, tuples are enclosed within parenthesis.
- The values stored in a tuple can be accessed using the slice operator ([ ] and [:])
- Indexes starting at 0 in the beginning of the tuple and working their way to end -1.
- The plus **(+)** sign is the concatenation operator.
- The asterisk **(*)** is the repetition operator.
- The main difference between lists and tuples are –
- Lists are enclosed in brackets ([ ]) and their elements and size can be changed,
- Tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as **read-only** lists.
- Python's dictionaries are kind of hash-table type.
- A dictionary consist of key-value pairs.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Python Operators:

- Python language supports the following types of operators –
    - Arithmetic Operators
    - Comparison (Relational) Operators
    - Assignment Operators
    - Logical Operators
    - Bitwise Operators
    - Membership Operators
    - Identity Operators

Operator	**Description**	**Example (a=10, b=21)**
+ Addition	Adds values on either side of the operator	a + b = 31

- Subtraction	Subtracts right hand operand from left hand operand	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 21 = 100000000000000000000
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true = False

!=	If values of two operands are not equal, then condition becomes true.	(a!= b) is true = True
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true = False
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true. = True
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true = False
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true = True

=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a

%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a
& Binary AND	Operator copies a bit, to the result, if it exists in both operands	(a & b) = 12 (means 0000 1100)
| Binary OR	It copies a bit, if it exists in either operand.	(a | b) = 61 (means 0011 1101)

^ Binary XOR	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011)
<< Binary Left Shift	Operator shifts the left operand bits towards the left side. The left side bits are removed. Binary number 0 is appended to the end.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	Exactly opposite to the left shift operator. The left operand bits are moved towards the right side, the right side bits are removed. Binary number 0 is appended to the beginning.	a >> 2 = 15 (means 0000 1111)
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False. (means 0)

or Logical OR	If any of the two operands are non-zero (true) then condition becomes true.	(a or b) is True. (means 1)
not Logical NOT	Used to reverse the logical state of its operand.	❖ Not (a and b) is True. (means True) ❖ Not (a or b) is False. (means False)
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1, if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1, if x is not a member of sequence y.
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1, if id(x) equals id(y).

is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1, if id(x) is not equal to id(y).

Operator precedence:

- The following table lists all operators from highest precedence to the lowest. Order - PEMDAS.

Operators	Meaning
()	**Parentheses**
****	**Exponent**
+x, -x, ~x	**Unary plus, Unary minus, Bitwise NOT**
***, /, //, %**	**Multiplication, Division, Floor division, Modulus**
+, -	**Addition, Subtraction**
<<, >>	**Bitwise shift operators**
&	**Bitwise AND**
^	**Bitwise XOR**
|	**Bitwise OR**
==, !=, >, >=, <, <=, is, is not, in, not in	**Comparisons, Identity, Membership operators**
not	**Logical NOT**
and	**Logical AND**
or	**Logical OR**

Decision Making:

- Decision Making statements are used to control the flow of execution of a program depending upon condition.
- Python programming language assumes any non-zero and non-null values as TRUE, and any zero or null values as FALSE value.
- Python programming language provides the following types of decision-making statements.
    - if statements
    - if…else statements
    - If …elif statements

if statement:

- The if statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

Syntax:

```
if expression:
    statement(s)
```

- If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.

if…else statement:

- An else statement can be combined with an if statement. An else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
- The else statement is an optional statement and there could be at the most only one else statement following if.

Syntax:

```
if expression:
    statement(s)
else:
    statement(s)
```

if…elif statement:

- The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the else, the elif statement is optional. However, unlike else, for which there can be at the most one statement, there can be an arbitrary number of elif statements following an if.

  Syntax:

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
```

```
        else:
            statememt(s)
```

nested if statement:

- When you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested if construct.
- In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct.

Syntax:

```
        if expression1:
            statement(s)
            if expression2:
                statement(s)
            elif expression3:
                statement(s)
            else:
                statememt(s)
        else:
            statement(s)
```

Looping:

- A loop statement allows us to execute a statement or group of statements multiple times.
- Python programming language provides the following types of loops to handle looping requirements.
    - while loop
    - for loop
    - nested loop

while loop:

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Syntax:

```
        while expression:
            statement(s)
```

- Here, statement(s) may be a single statement or a block of statements with uniform indent.
- Python supports having an else statement associated with a loop statement.
- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

for loop:

- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax:

```
for iterating_var in sequence:
    statement(s)
```

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.
- The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.
- Python supports having an else statement associated with a loop statement.
- If the else statement is used with a for loop, the else block is executed only if for loops terminates normally (and not by encountering break statement).

nested loop:

- Python programming language allows the usage of one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax:

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s)


while expression:
    while expression:
        statement(s)
            statement(s)
```

Loop control statements:

- The Loop control statements change the execution from its normal sequence.
- When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements.
    - break statement
    - continue statement
    - pass statement

break statement:

- Terminates the loop statement and transfers execution to the statement immediately following the loop.
- The break statement stops the execution of the innermost loop and starts executing the next line of the code after the block.

Syntax:

```
break
```

continue statement:

- The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

Syntax:

continue

pass statement:

- The pass statement is a *null* operation; nothing happens when it executes.
  The pass statement is also useful in places where your code will eventually go, but has not been written yet i.e. in stubs.

Syntax:

pass

****

Prepared By,
R.Chithra Devi
Asso.Prof / IT
Dr.SACOE
Tiruchendur.