Way To Python

CodeHub Academy

**VARIABLES AND DATA TYPES**

**1. What is a variable ?**

- Variables are reserved memory locations that are used to store values and are referenced by a name.

Example: Assigning a contact number and referencing it by a contact name

**2. How to define a variable?**

- Syntax to define a variable is as follows:

variableName = value

Example: phoneNumber = 12345

**3. What are the rules for declaring a variable?**

- Variable name should start with an alphabet or underscore (‘\_’) followed by any number of alphabets, digits and underscores

a. Variable names are case sensitive.

Example: phoneNumber is different from PhoneNumber and phonenumber

b. Variable name cannot be a reserved name

Example: print, if, for, etc

c. Variable name cannot contain any special character other than underscore

**4. What are the different data types in Python ?**

In Python, data types are broadly classified into the following:

1. Numbers

2. List

3. Tuple

4. Strings

5. Dictionary

**NUMBERS**

**1. What is number ?**

A number is a mathematical object used to count, measure and label.

**2. What are the different types of numbers in Python ?**

**1. Integer numbers**

Any non decimal number is called an integer

number.

Example: 10, -5000

**2. Floating point numbers**

Any decimal number is called a floating point   
 number.

Example: 5.5, -69.3765

**OPERATORS**

**1. What is an operator ?**

An operator is an action performed between two operands

Example: 2 + 3 where ‘+’ is the operator and 2 and 3 are the operands. Addition is the action performed in this operation.

**2. What are the different types of operators ?**

Operators are broadly classified into the following:

**1. Assignment Operator** (‘=‘) => Assigns the value at the right hand side to the variable at the left hand side

**2. Arithmetic Operators**

- Multiplication (‘\*’) => Returns the product of two operands. Example: 3 \* 3 returns 9

- Division (‘/‘) => Returns the quotient of two operands. Example: 100/10 returns 10

- Addition (‘+’)=> Returns the sum of two operands. Example: 2 + 2 returns 4

- Subtraction (‘-‘) => Returns the difference between two operands. Example: 10 - 7 return 3

- Modulo (‘%’)=> Returns the remainder from the division performed between two operands. Example: 10 % 7 returns 3

**3. Relational or Comparison Operators**

- Equal to (‘==‘) => Returns true if both the operands are equal. Example: (2 == 2)

- Greater than (‘>’) => Returns true if operand 1 is greater than operand 2. Example: (1 > 2)

- Lesser than (‘<‘) => Returns true if operand 1 is lesser than operand 2.

Example: (2 > 1)

- Greater than or equal to (‘>=’) => Returns true if operand 1 is greater than or equal to operand 2.

Example: (2 >= 2)

- Lesser than or equal to (‘<=‘) => Returns true if operand 1 is lesser than or equal to operand 2.

Example: (2 <= 2)

- Not equal to (!=) => Returns true if operand 1 is not equal to operand 2.

Example (2 != 3)

**4. Logical Operators**

- and => Returns true only if both the operands are true.

Example: ((5 > 3) and (3 < 5))

- or => Returns true if at least among the operands is true.

Example: ((5 < 3) or (3 < 5))

- not => Reverses the boolean value of the operand.

Example not(5<3) returns True (reverse of False).

**STRING**

# What is a string ?

A string is a sequence of characters that are enclosed within single quotes or double quotes.

Example: “Python is a very simple language.” or ‘Python is a very simple language’

# How to define a string ?

Syntax to define a string is as follows: stringName = “string”

or

stringName = ‘string’

Example: programmingLanguage = “Python”

# How are strings stored ?

Strings are stored as an indexed sequence of characters.

The starting index is zero.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **String** | P | Y | T | H | O | N |
| **Index** | 0 | 1 | 2 | 3 | 4 | 5 |

# What are some of the string operations ?

Following are some of the string operations that can be performed in Python. Consider the following example string which we will be using for further explanation.

welcomeMessage = “Welcome to this Introduction course on Python programming. Hope you are having a good time learning.”

# Upper

Converts all the characters of a string to uppercase. Syntax: stringName.upper()

Example: “welcomeMessage.upper()”

returns 'WELCOME TO THIS INTRODUCTION COURSE ON

PYTHON PROGRAMMING. HOPE YOU ARE HAVING A GOOD TIME LEARNING.'

# Lower

Converts all the characters of a string to lowercase. Syntax: stringName.lower()

Example: “welcomeMessage.lower()” returns 'welcome to this introduction course on python programming. hope you are having a good time learning.'

# Replace

Replaces the specified sequence of character(s) in the first parameter with that of the second parameter.

Syntax: stringName.replace(“sequenceToBeReplaced”, “sequenceToReplace”)

Example: “welcomeMessage.replace(“learning”, “experimenting”)” returns 'welcome to this introduction course on python programming. hope you are having a good time experimenting.'

**d. Slicing**

Returns a slice of the string based on the indices passed.

Syntax: stringName[startingIndex:endingIndex]

Note:

1. Characters starting from index “startingIndex” upto “endingIndex - 1” will be printed

Example: welcomeMessage[0:7] returns “welcome”

2. If startingIndex is not mentioned, characters starting from 0 to “endingIndex - 1” will be printed

Example: welcomeMessage[:57] returns 'welcome to this introduction course on python programming'

3. If endingIndex is not mentioned, characters starting from startingIndex to end of the string will be printed

Example: welcomeMessage[59:] returns 'hope you are having a good time experimenting.'

1. If both startingIndex and endingIndex are not mentioned, the whole string will be printed

Example: welcomeMessage[:] returns 'welcome to this introduction course on python programming. hope you are having a good time experimenting.'

# Length

Returns length of a string Syntax: len(string)

Example: len(welcomeMessage) returns 100

**5. Basic string formatting in python** Consider  
the following string. programmingLanguage = “Python”

If you need to include this string as a part of another string,

you need to use a set of opening and closing curly braces as a placeholder ‘{}’

Example: print “Welcome to {} programming”.format(programmingLanguage)

This would print “Welcome to Python programming.”

**LIST**

**1. What is a list ?**

A list is a container that holds many objects under a single name.

**2. How to define a list ?**

Syntax to define a list is as follows:

listName = [object1, object2, object3]

Example: bestFriends = ['Mark', 'Mary', 'Maria', 'John’]

**3. How to access the values in a list ?**

The values in a list can be accessed by specifying the index of that value. Values are indexed starting from zero. From the bestFriends list, values can be accessed as mentioned below:

bestFriends[0] returns ‘Mark’

bestFriends[1] returns ‘Mary’

bestFriends[2] returns ‘Maria’

bestFriends[3] returns ‘John’

bestFriends[4] returns an index out of range error as our list contains only four elements

**4. What are some of the list operations ?**

**1. Append**

Append operation is used to add a new element to end of the list.

Syntax: list.append(element)

Example: bestFriends.append(“Febin”)

bestFriends list now returns the list with the name Febin appended to the end of the list

['Mark', 'Mary', 'Maria', 'John’, 'Febin']

**2. Insert**

Insert operation is used to add a new element at a specified index and shift the other elements to the right.

Syntax: list.insert(index, element)

Example: bestFriends.insert(1, “Ben”)

bestFriends list now returns the previous list along with name Ben placed at index 1

['Mark', 'Ben', 'Mary', 'Maria', 'John’, 'Febin']

**3. Remove**

Remove operation is used to remove an element from the list

Syntax: list.remove(element)

Example: bestFriends.remove(“Mary”)

bestFriends list now returns the list without the name Mary

['Mark', 'Ben', 'Maria', 'John’, 'Febin']

**4. Sort**

Sort operation is used to sort a given list in ascending order.

Syntax: list.sort()

Example: bestFriends.sort()

bestFriends list now gets sorted alphabetically as

['Ben', 'Febin', 'John’, 'Maria', 'Mark']

**5. Reverse**

Reverse operation is used to reverse a list.

Syntax: list.reverse()

Example: bestFriends.reverse()

bestFriends list now returns the list in reverse order with the name Ben now being at the end of the list

['Mark', 'Maria', 'John’, 'Febin', 'Ben']

**6. Pop**

Pop operation is used to return an element at the specified index and remove it from the list.

Syntax: list.pop(index)

Example: bestFriends.pop(2) returns ‘John’

bestFriends list now the list without the name John in it

['Mark', 'Maria', 'Febin', 'Ben']

If no index is specified, pop operation returns the last element of the list.

Example: bestFriends.pop() returns ‘Ben’

bestFriends list now returns the list with the name Ben removed from the end of the list

['Mark', 'Maria', 'Febin']

**DICTIONARY**

**1. What is a dictionary in Python ?**

A dictionary is a set of key-value pairs referenced by a single name

**2. How to create a dictionary ?**

The syntax to create a dictionary is as follows:

dictionaryName = {“keyOne” : “valueOne”, “keyTwo”, “valueTwo”}

Example: Consider the following dictionary that stores the colour of fruits with key as the fruit name as value as its color.

colorOfFruits = {“apple”: “red”, “mango”: “yellow”, “orange”: “orange”}

**3. How to retrieve values from a dictionary ?**

Values are retrieved from a dictionary by specifying the key associated to that value.

Syntax: dictionaryName[“key”]

Example: To retrieve the color of mango, type the following line of code

colorOfFruits[“mango”]

This will return 'yellow'

**4. How to update a value associated to a dictionary key ?**

A value can be reassigned by making use of the key corresponding to that value.

Syntax: dictionaryName[“key”] = “New Value”

Example: colorOfFruits[“apple”] = “green”

Our dictionary now becomes {'orange': 'orange', 'mango': 'yellow', 'apple': 'green'}

**5. What are some of the operations of a dictionary ?**

1. List all keys

keys() is used to list all the keys in a dictionary.

Syntax: dictionaryName.keys()

Example: colorOfFruits.keys() returns ['orange', 'mango', 'apple']

2. List all values

values() is used to list all the values in a dictionary

Syntax: dictionaryName.values()

Example: colorOfFruits.values() returns ['orange', 'yellow', 'green']

**3. Delete a key-value pair**

del keyword is used to delete a key-value pair from a dictionary

Syntax: del dictionaryName[“key”]

Example: del colorOfFruits[“apple”]. Our dictionary now returns {'orange': 'orange', 'mango': 'yellow'}

**4. Copy a dictionary into another**

copy() is used to copy the contents of one dictionary to another

Syntax: dictionaryTwo = dictionaryOne.copy()

Example: copyOfColorOfFruits = colorOfFruits.copy()

copyOfColorOfFruits will now return {'orange': 'orange', 'mango': 'yellow'}

**5. Clear a dictionary**

clear() is used to clear the contents of a dictionary and make it empty

Example: colorOfFruits.clear()

colorOfFruits now returns {}, which is an empty dictionary

**CONDITIONALS**

**1. What are conditional statements ?**

Condition statements are a block of statements whose execution depends on a certain condition.

**2. What are the different types of conditional statements in Python ?**

1. If:

A “simple if” condition is one where a block of statements get executed if the condition mentioned in the “if” statement evaluates to true

Example:

distance = 100

if distance == 100:

print(“Distance is 100”)

**2. If-Else:**

An “If-Else” statement is one where a block of statements under “if” condition gets executed if the condition evaluates to true. If the condition evaluates to false, the block of statements under “else” is executed.

Example:

distance = 200

if distance <= 100:

print(“Distance is less than or equal to 100”)

else:

print(“Distance is greater than 100”)

**3. If-Elif-Else**

An “If-Elif-Else” statement is one where multiple “if” conditions are evaluated one after another if an “if” statement evaluates to false. “elif” stands for else-if. If all the if conditions evaluates to false, the block of statements under “else” gets executed.

Example:

distance = 400

if distance <= 100:

print(“Distance is less than or equal to 100”)

elif distance <= 200:

print(“Distance is less than or equal to 200”)

elif distance <= 300:

print(“Distance is 300”)

else:

print(“Distance is greater than 300”)

**4. Nested If**

An if statement within another if statement is called a nested if statement.

Example:

distance = 50

if distance < 100:

if distance == 50:

print “Distance is 50”

**TUPLE**

**1. What is a tuple ?**

A tuple is a container that holds many objects under a single name. A tuple is immutable which means, a tuple once defined cannot be modified.

**2. How to define a tuple ?**

Syntax to define a tuple is as follows:

tupleName = (object1, object2, object3)

Example: dateOfBirth = (“01-01-1900”, “31-12-2016”)

**3. How to access the values in a tuple ?**

The values in a tuple can be accessed by specifying the index of that value. Values are indexed starting from zero.

Example: dateOfBirth[1] returns 31-12-2016

**4. How to delete a tuple ?**

A tuple can be deleted using the del keyword

Syntax: del(tupleName)

Example: del(dateOfBirth) deletes our tuple dateOfBirth and can no longer be accessed

**LOOPS**

**1. Why is looping used ?**

Looping is used to repeatedly perform a block of statements over and over again.

**2. What are the different types of loops in Python ?**

**1. For Loop:**

For loop is used to iterate over a sequence, starting from the first value to the last. The number of iterations to be performed depends upon the length of the list.

Syntax:

for iteratingVariable in sequence:

statement 1

statement 2

- - -

- - -

statement n

Example:

numbers = [1, 2, 3, 4, 5]

for number in numbers:

print(number, end=‘ ‘)

Output:

1 2 3 4 5

Note: end = ‘ ’ is used to end the print statement with a white space instead of a new line

In the example mentioned above, the variable “number” is used to iterate over the list “numbers”. Here, the loop is executed 5 times since the length of the list is 5. At first iteration, the variable “number” holds the value “1”, at the second iteration value it holds the value “2” and so on.

**2. While Loop:**

“While loop” is used to repeatedly execute a block of statements as long as the condition mentioned in the “while loop” holds true.

Syntax:

while condition:

statement 1

statement 2

- - -

- - -

statement 3

Example:

length = 1

while length <= 3:

print(“Value of length = “, length)

length = length + 1

Output:

Value of length = 1

Value of length = 2

Value of length = 3

In this example, the execution of while loop depends on the value stored in the variable “length”. Each time the block of code in while loop gets executed, we increment the value of “length” by 1. When the value stored in the variable length is “4”, the condition “length <= 3” turns false and the loop breaks.

**3. Nested Loop:**

A loop within another loop is called a nested loop. The concept of nested loop could be a little bit of trouble understanding at first, but can be simplified with the help of an example.

outerLoopValue = 1

innerLoopValue = 1

while outerLoopValue <= 2:

# This inner loop runs three times for every iteration of the outer loop

while innerLoopValue <=3:

print(“Outer loop value = “, outerLoopValue)

print(“Inner loop value = “, innerLoopValue)

# Increment the inner loop iteration value

innerLoopValue = innerLoopValue + 1

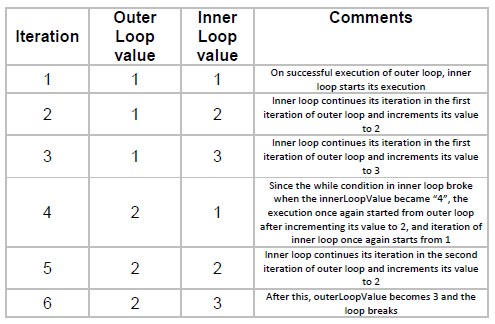
# Increment the outer loop iteration value

outerLoopValue = outerLoopValue + 1

# Reset the inner loop value to 1

innerLoopValue = 1

In a nested loop, only after the completion of all iterations of the innermost loop does the outer loop proceed to its next iteration. Please refer the table below with reference to the above example.



3. What are break, continue and else statements ?

Break:

A break statement is used to stop a loop from further execution.

Example:

length = 1

while length > 0:

if length == 3:

break

print(“Length = “, length)

length = length + 1

Output:

Length = 1

Length = 2

Length = 3

In the above example, when length = 3, the break statement gets executed and the while loop breaks.

**Continue:**

Continue statement is used to skip a particular iteration of the loop.

Example:

length = 1

while length <= 4:

if length == 2:

length = length + 1

continue

print(“Length = “, length)

length = length + 1

Output:

Length = 1

Length = 3

Length = 4

In the above example, when length = 2, continue statement stops further execution of that iteration and moves on to the next iteration

**Note: While break statement stops the whole loop from execution, continue stops just an iteration of that loop.**

Else:

The block of statements in the else block gets executed if the break statement in the looping condition was executed

Example:

length = 1

while length <= 3:

if length == 5:

break

print(“Length = “, length)

length = length + 1

else:

print(“Break statement was not executed”)

Output:

Length = 1

Length = 2

Length = 3

Break statement was not executed

In the above example, the break statement does not get executed and the statement in else block gets executed after the loop.