Introduction to Python

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

27 March 2021

Overview

- **Python** is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- It was created by Guido van Rossum during 1985- 1990.
- Like Perl, Python source code is also available under the GNU General Public License (GPL).
- Python is designed to be highly readable.
- It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Advantages of Learning Python

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Applications of Python

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

Hello, World program in Python

- Example:
  print(“Hello, world!”)
- Compared to C programming, where

```
#include<stdio.h>
int main() {
        printf(“Hello, world!”);
}
```

Environment Setup

- Available for Windows, Linux-based systems and MacOS.
- Up-to-date binaries available on python.org.
- Documentation for Python also available on python.org.
- Linux Installation – download and extract source code, configure script, make and install.
- Windows Installation – download and install using installer setup file.
- MacOS installation – download and install using installer setup file.
- Setup PATH environment variable before running from terminal/command prompt/IDE.

Python Programming

- **Interactive shell programming** – interaction with Python shell/terminal/command line.
  Example:
  $ python
  >>> print(“Hello, world!”)
  >>> Hello, world!
- **Script mode programming** – write the code in a script, save it as *filename.py* and run the code in the terminal/command line using:
  python filename.py

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, **Hello1** and **hello1** are two different identifiers in Python.

Python Naming Conventions

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Python Reserved Words

- Keywords are reserved words and you cannot use them as constants or variables or any other identifier names.
- All the Python keywords contain lowercase letters only.

and	**exec**	**not**
assert	**finally**	**or**
break	**for**	**pass**
class	**from**	**print**
continue	**global**	**raise**
def	**if**	**return**
del	**import**	**try**
elif	**in**	**while**
else	**is**	**with**
except	**lambda**	**yield**

Python Indentation

- Python provides no 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.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- For example –

```
if True:
        print "True“
else:
        print "False"
```

Python Multi-line Statements

- Statements in Python typically end with a new line.
- Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.
- For example –

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character.
- For example –

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
        'Friday']
```

Python Quotation

- 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.
- For example, all the following are legal –
  - **word = 'word‘**
  - **sentence = "This is a sentence."**
  - **paragraph = """This is a paragraph. It is made up of multiple lines and sentences."""**

Python Comments

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
- Example:
  #!/usr/bin/python
  # First comment
  print "Hello, Python!" # second comment
- Triple-quoted string is also ignored by Python interpreter and can be used as a multiline comment.

Variables

- Variables are nothing but reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Variables

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.
- Example:

```
#!/usr/bin/python
counter = 100      # An integer assignment
miles = 1000.0     # A floating point
name = "John"      # A string
print counter
print miles
print name
```

Variables

- **Multiple assignment -**
- Python allows you to assign a single value to several variables simultaneously.
- For example –
  a = b = c = 1
  Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables.
- For example –
  a,b,c = 1,2,“john”
  Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Data Types

- The data stored in memory can be of many types.
- For example, a person's age is stored as a numeric value and his/her address is stored as alphanumeric characters.
- 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 data types –
  - Number
  - String
  - List
  - Tuple
  - Dictionary

Operators

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

Operators - Membership

- Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.

Operator	Description	Example
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.

Operators - Identity

- Python's identity operators compare the memory locations of two objects.

Operator	Description	Example
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).

Operators - Identity

- Python's identity operators compare the memory locations of two objects.

Operator	Description	Example
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).

Decision Making – if Statement

- The **if** statement contains a logical expression using which 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 Boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

Decision Making – if Statement

- Example:

```
#!/usr/bin/python
var1 = 100
if var1:
      print "1 - Got a true expression value“
      print var1
var2 = 0
if var2:
      print "2 - Got a true expression value“
      print var2
print "Good bye!"
```

Decision Making – if…else Statement

- An **else** statement can be combined with an **if** statement.
- An **else** statement contains the 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 most only one **else** statement following **if**.
- Syntax:

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

Decision Making – if…else Statement

- Example:

```
#!/usr/bin/python
var1 = 100
if var1:
      print "1 - Got a true expression value“
      print var1
else:
      print "1 - Got a false expression value“
      print var1
var2 = 0
if var2:
      print "2 - Got a true expression value“
      print var2
else:
      print "2 - Got a false expression value“
      print var2
print "Good bye!"
```

Decision Making – nested if Statement

- There may be a situation 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:
            statement(s)
else:
        statement(s)
```

Decision Making – nested if Statement

- Example:

```
#!/usr/bin/python
var = 100
if var < 200:
      print "Expression value is less than 200“
      if var == 150:
          print "Which is 150“
      elif var == 100:
          print "Which is 100“
      elif var == 50:
          print "Which is 50“
      elif var < 50:
          print "Expression value is less than 50“
else:
      print "Could not find true expression"
```

Thank you!