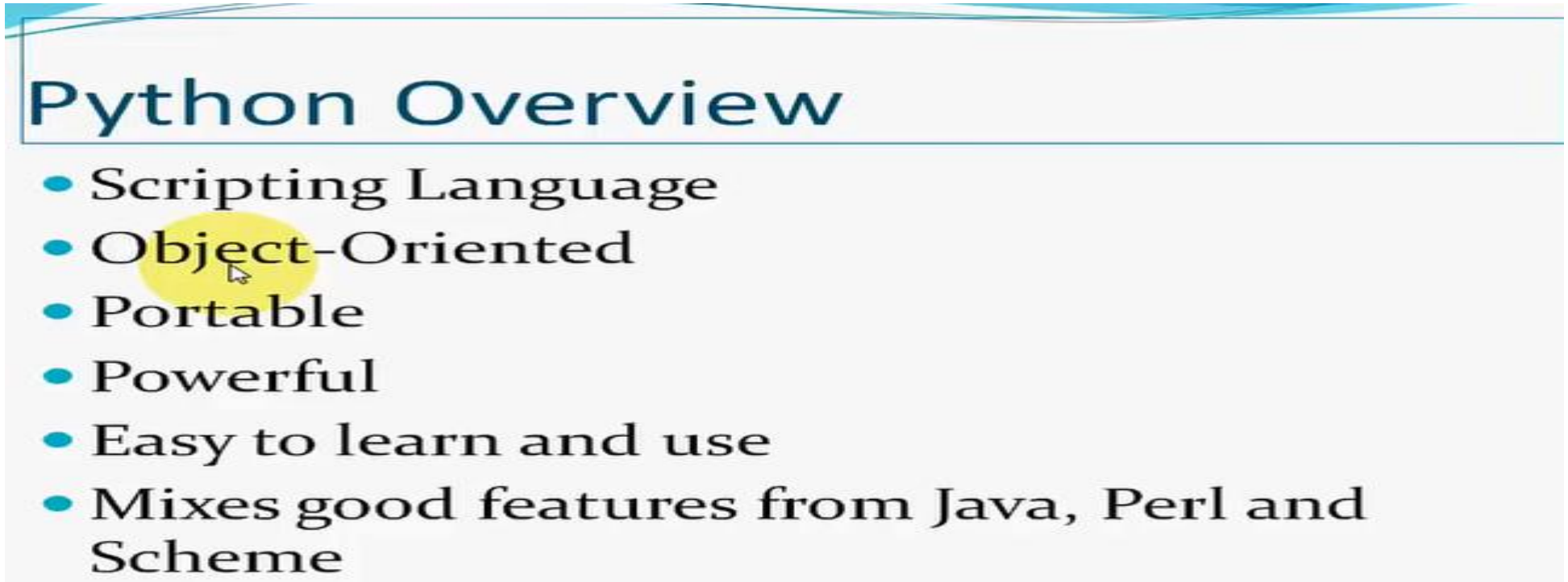
Python

RAVINDRA KUDACHE

Python Overview



Python Overview

Major Uses of Python

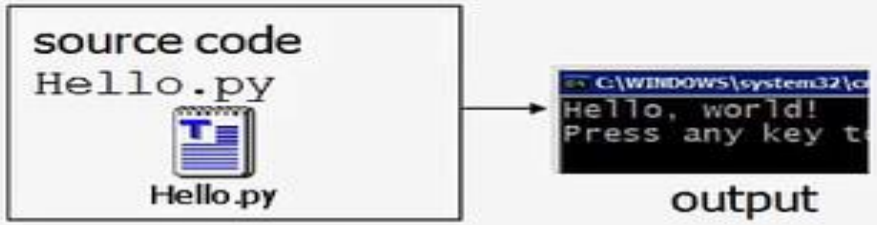
- System Utilities
- GUIs (Tkinter, gtk, Qt, Windows)
- Internet Scripting
- Embedded Scripting
- Database Programming
- Artificial Intelligence
- Image Processing

Python Overview

Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.

- Python is instead directly *interpreted* into machine instructions.



Python Overview

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 b
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print hi
SyntaxError: Missing parentheses in call to 'print'
>>> print ("hi")
hi
>>>
```

```
[root@localhost ~]# python
Python 2.4.3 (#1, Jun 11 2009, 14:09:58)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-44)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "I am 2.4 version i am printable without brackrt"
I am 2.4 version i am printable without brackrt
>>>
```

Python Overview

- Python is a high-level, interpreted, interactive and object-oriented scripting language. 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.
- **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.

History of Python

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

- **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.
- Apart from the above-mentioned features, Python has a big list of good features, few are listed below:
- 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.

Installing Python

- To download and install Python visit the official website of Python http://www.python.org/downloads/ and choose your version. We have chosen Python version 2.7.5
- **Step 2: Choosing an editor/IDE**

To run your Python, you need an editor. There some popular online editor for Python language that you can use like

http://pythonfiddle.com

http://sublimetext.com

http://www.aptana.com/products/studio3

Step follow for linux python installation

1)Download and extract

wget http://www.python.org/downloads/ftp/python/3.6/Python-3.6.0.tar.xz

tar xjf Python-3.6.0.tar.xz

If file in Python-2.7.13 (1).tgz format

gunzip Python-2.7.13 (1).tgz

tar –xvf Python-2.7.13 (1).tar

2) ./configure

3)make

4) make install

Installing Python

Implementations of Python:-

- CPython CPython is the reference implementation of Python, written in C. It compiles Python code to intermediate bytecode which is then interpreted by a virtual machine. CPython provides the highest level of compatibility with Python packages and C extension modules. If you are writing open-source Python code and want to reach the widest possible audience, targeting CPython is best. To use packages which rely on C extensions to function, CPython is your only implementation option. All versions of the Python language are implemented in C because CPython is the reference implementation
- PyPy PyPy is a Python interpreter implemented in a restricted statically-typed subset of the Python language called RPython. The interpreter features a just-in-time compiler and supports multiple back-ends (C, CLI, JVM). PyPy aims for maximum compatibility with the reference CPython implementation while improving performance. If you are looking to increase performance of your Python code, it's worth giving PyPy a try. On a suite of benchmarks, it's currently over 5 times faster than CPython. PyPy supports Python 2.7. PyPy3 1 , released in beta, targets Python 3
- Jython Jython is a Python implementation that compiles Python code to Java bytecode which is then executed by the JVM (Java Virtual Machine). Additionally, it is able to import and use any Java class like a Python module. If you need to interface with an existing Java codebase or have other reasons to need to write Python code for the JVM, Jython is the best choice. Jython currently supports up to Python 2.7. 2
- IronPython IronPython is an implementation of Python for the .NET framework. It can use both Python and .NET framework libraries, and can also expose Python code to other languages in the .NET framework. Python Tools for Visual Studio integrates IronPython directly into the Visual Studio development environment, making it an ideal choice for Windows developers. IronPython supports Python 2.7. 3
- PythonNet Python for .NET is a package which provides near seamless integration of a natively installed Python installation with the .NET Common Language Runtime (CLR). This is the inverse approach to that taken by IronPython (see above), to which it is more complementary than competing with. In conjunction with Mono, pythonnet enables native Python installations on non-Windows operating systems, such

Python keywords

We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

There are 33 keywords in Python 3.3. This number can vary slightly in course of time.

All the keywords except `True`, `False` and `None` are in lowercase and they must be written as it is. The list of all the keywords are given below.

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

Python Identifiers and rules of writing

Python Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for writing identifiers

1. 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 (_). Names like `myClass`, `var_1` and `print_this_to_screen`, all are valid example.
2. An identifier cannot start with a digit. `1variable` is invalid, but `variable1` is perfectly fine.
3. Keywords cannot be used as identifiers.

```
>>> global = 1
  File "<interactive input>", line 1
    global = 1
           ^
SyntaxError: invalid syntax
```

4. We cannot use special symbols like !, @, #, $, % etc. in our identifier.

```
>>> a@ = 0
  File "<interactive input>", line 1
    a@ = 0
     ^
SyntaxError: invalid syntax
```

5. Identifier can be of any length.

Things to care about

Python is a case-sensitive language. This means, `Variable` and `variable` are not the same. Always name identifiers that make sense.

While, `c = 10` is valid. Writing `count = 10` would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, `this_is_a_long_variable`.

We can also use camel-case style of writing, i.e., capitalize every first letter of the word except the initial word without any spaces. For example: `camelCaseExample`

Python Statement, Comments

Instructions that a Python interpreter can execute are called statements. For example, `a = 1` is an assignment statement. `if` statement, `for` statement, `while` statement etc. are other kinds of statements which will be discussed later.

Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
colors = ['red',
          'blue',
          'green']
```

We could also put multiple statements in a single line using semicolons, as follows

```
a = 1; b = 2; c = 3
```

Python Statement, Comments

2.4 Script mode

So far we have run Python in **interactive mode**, which means that you interact directly with the interpreter. Interactive mode is a good way to get started, but if you are working with more than a few lines of code, it can be clumsy.

The alternative is to save code in a file called a **script** and then run the interpreter in **script mode** to execute the script. By convention, Python scripts have names that end with `.py`.

If you know how to create and run a script on your computer, you are ready to go. Otherwise I recommend using PythonAnywhere again. I have posted instructions for running in script mode at `http://tinyurl.com/thinkpython2e`.

Because Python provides both modes, you can test bits of code in interactive mode before you put them in a script. But there are differences between interactive mode and script mode that can be confusing.

For example, if you are using Python as a calculator, you might type

```
>>> miles = 26.2
>>> miles * 1.61
```

The first line assigns a value to `miles`, but it has no visible effect. The second line is an expression, so the interpreter evaluates it and displays the result. It turns out that a marathon is about 42 kilometers.

But if you type the same code into a script and run it, you get no output at all. In script mode an expression, all by itself, has no visible effect. Python actually evaluates the expression, but it doesn't display the value unless you tell it to:

```
miles = 26.2
print(miles * 1.61)
```

This behavior can be confusing at first.

A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script

```
print(1)
x = 2
print(x)
```

produces the output

```
1
2
```

Python Indentation

- Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
- A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.
- Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.
- The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.
- Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

Python Comments

- **Multi-line comments**
- “”” or ‘’’ for comment on multiple line

E.g:- """This is also a

perfect example of

multi-line comments"""

Docstring in Python

- Docstring is short for documentation string.
- It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
- Triple quotes are used while writing docstrings. For example:

```
def double(num):
    """Function to double the value"""
        return 2*num
```

Docstring in Python

Lines and 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:

However, the following block generates an error:

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks:

Note: Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

Docstring in 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']
```

Docment string in Python

Quotation in Python

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."""
```

Thank You