Features of Python

**Open source:**Python is publicly available open source software, any one can use source code that doesn't cost anything.

**Easy-to-learn:** Popular (scripting/extension) language, clear and easy syntax, no type declarations, automatic memory management, high-level data types and operations, design to read (more English like syntax) and write (shorter code compared to C, C++, and Java) fast.

**High-level Language:**   
High-level language (closer to human) refers to the higher level of concept from machine language (for example assembly languages). Python is an example of a high-level language like C, C++, Perl, and Java with low-level optimization.

**Portable:**  
High level languages are portable, which means they are able to run across all major hardware and software platforms with few or no change in source code. Python is portable and can be used on Linux, Windows, Macintosh, Solaris, FreeBSD, OS/2, Amiga, AROS, AS/400 and many more.

**Object-Oriented:** Python is a full-featured object-oriented programming language, with features such as classes, inheritance, objects, and overloading.

**Python is Interactive :**  
Python has an interactive console where you get a Python prompt (command line) and interact with the interpreter directly to write and test your programs. This is useful for mathematical programming.

**Interpreted :** Python programs are interpreted, takes source code as input, and then compiles (to portable byte-code) each statement and executes it immediately. No need to compiling or linking

**Extendable :** Python is often referred to as a "glue" language, meaning that it is capable to work in mixed-language environment. The Python interpreter is easily extended and can add a new built-in function or modules written in C/C++/Java code.

**Libraries :** Databases, web services, networking, numerical packages, graphical user interfaces, 3D graphics, others.

**Supports :**Support from online Python community

Python Interpreter

* In interactive mode, type Python programs and the interpreter displays the result:
* Type python into your terminal's command line
* After a short message, the >>> symbol will appear
* The above symbol signals the start of a Python interpreter's command line.
* Python interpreter evaluates inputs (For example >>> 4\*(6-2) return 16)

How stable is Python?

Very stable. New, stable releases have been coming out roughly every 6 to 18 months since 1991, and this seems likely to continue. Currently there are usually around 18 months between major releases.

The latest stable releases can always be found on the [Python download page](https://www.python.org/download/). There are two recommended production-ready versions at this point in time, because at the moment there are two branches of stable releases: 2.x and 3.x. Python 3.x may be less useful than 2.x, since currently there is more third party software available for Python 2 than for Python 3. Python 2 code will generally not run unchanged in Python 3.

History

The name Python was selected from "Monty Python�s Flying Circus" which was a British sketch comedy series created by the comedy group Monty Python and broadcast by the BBC from 1969 to 1974.

Python was created in the early 1980s by Guido van Rossum at the National Research Institute for Mathematics and Computer Science in Netherlands.

Python was created as a successor of a language called ABC (All Basic Code) and released publicly in1991. Guido remains Python's principal author, although it includes many contributions from active user community.

Between 1991 and 2001 there are several versions released, current stable release is 3.2. In 2001 the Python Software Foundation (PSF) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

Major uses of Python

* System utilities (system admin tools, command line programs).
* Web Development.
* Graphical User Interfaces (Tkinter, gtk, Qt).
* Internet scripting.
* Embedded scripting.
* Database access and programming.
* Game programming.
* Rapid prototyping and development.
* Distributed programming

Organizations Using Python (sector wise)

**Web Development :** Yahoo Maps, Yahoo Groups, Google, Zope Corporation, Ultraseek, Linux Weekly News, ElasticHosts Cloud Servers, Mojam.com, hunch, Shopzilla, Movieplayer.it, Multiplayer.it.

**Games:**Battlefield 2, Crystal Space, Star Trek Bridge Commander, The Temple of Elemental Evil, Vampire: The Masquerade: Bloodlines, Civilization 4, QuArK (Quake Army Knife)

**Graphics :**Industrial Light & Magic, Walt Disney Feature Animation, HKS, Inc. (ABAQUS/CAE), RoboFog, Caligari Corporation, Blender 3D, Jasc Software, Paint Shop Pro.

**Financial :** Altis Investment Management, ABN AMRO Bank, Treasury Systems, Bellco Credit Union, Journyx Timesheet and Resource Management Software.

**Science :** National Weather Service, Radar Remote Sensing Group, Applied Maths, Biosoft, The National Research Council of Canada, Los Alamos National Laboratory (LANL) Theoretical Physics Division, AlphaGene, Inc., LLNL, NASA, Swedish Meteorological and Hydrological Institute (SMHI), Environmental Systems Research Institute (ESRI), Objexx Engineering, Nmag Computational Micromagnetics

**Electronic Design Automation:**Ciranova, Productivity Design Tools, Object Domain, Pardus, Red Hat, SGI, Inc., MCI Worldcom, Nokia,

**Education** : University of California, Irvine, Smeal College of Business, The Pennsylvania State University, New Zealand Digital Library, IT Certification Exam preparation, SchoolTool,

**Business Software :**Raven Bear Systems Corporation, Thawte Consulting, Advanced Management Solutions Inc., IBM, Arakn<E9>, RealNetworks, dSPACE, Escom, The Tiny Company, Nexedi, Piensa Technologies - Bufete Consultor de Mexico, Nektra, WuBook.

To see the details of the above organizations [check here](https://wiki.python.org/moin/OrganizationsUsingPython).

Is Python a good language for beginning programmers?

Yes. It is still common to start students with a procedural and statically typed language such as Pascal, C, or a subset of C++ or Java. Students may be better served by learning Python as their first language. Python has a very simple and consistent syntax and a large standard library and, most importantly, using Python in a beginning programming course lets students concentrate on important programming skills such as problem decomposition and data type design. With Python, students can be quickly introduced to basic concepts such as loops and procedures. They can probably even work with user-defined objects in their very first course.

For a student who has never programmed before, using a statically typed language seems unnatural. It presents additional complexity that the student must master and slows the pace of the course. The students are trying to learn to think like a computer, decompose problems, design consistent interfaces, and encapsulate data. While learning to use a statically typed language is important in the long term, it is not necessarily the best topic to address in the students’ first programming course.

Many other aspects of Python make it a good first language. Like Java, Python has a large standard library so that students can be assigned programming projects very early in the course that do something. Assignments aren’t restricted to the standard four-function calculator and check balancing programs. By using the standard library, students can gain the satisfaction of working on realistic applications as they learn the fundamentals of programming. Using the standard library also teaches students about code reuse. Third-party modules such as PyGame are also helpful in extending the students’ reach.

## Comparison between Python 2 and Python 3

The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release. Python 3.0 was released in 2008. The 2.x release will see no new major releases after that. 3.x is under active development and has already seen over several years of stable releases, including version 3.3 in 2012, 3.4 in 2014, and 3.5 in 2015.

If you're new to programming or an experienced developer, it's easy to learn and use Python 3. Python 3 represents the future of the language as all development has stopped with Python 2.x, save for security and bugfixes.

print statement vs. print function

In Python 3 the print statement has been replaced with a print() function, with keyword arguments to replace most of the special syntax of the old print statement.

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> print "Hello World"  Hello World | >>> print("Hello World")  Hello World |
| >>> #print a newline  >>> print | >>> #print a newline,  >>> #using parentheses  >>> print() |
| >>> #add trailing comma to  >>> #remove newline  >>> print "No newline",  No newline | >>> #Appends nothing  >>> #instead of a newline  >>> print("No newline", end="")  No newline  >>> |
| >>> #print to stderr  >>> print >>sys.stderr, "Error"  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  NameError: name 'sys' is not defined | >>> #Specifies the  >>> #output buffer  >>>print("Error", file=sys.stderr)  >>>  Traceback (most recent call last):  File "C:/Python34/test.py",  line 1, in <module>  print("Error", file=sys.stderr)  NameError: name 'sys'  is not defined  >>> |
|  | >>> #sep specifies the separator  >>> #here ',' is used as a separator  >>> print("Python", "JS", sep=",")  Python,JS |
|  | >>> #prints as XYZ, there  >>> #is nothing in sep  >>> print("X", "Y", "Z", sep="")  XYZ |
|  | >>> #flush the output buffer  >>> print("Python", flush=True)  Python |

Differences between range and xrange functions

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> range(1, 10)  [1, 2, 3, 4, 5, 6, 7, 8, 9] | >>> range(1, 10)  range(1, 10)  >>> |
| >>> xrange(1, 10)  xrange(1, 15) | >>> xrange(1, 10)  Traceback (most recent call last):  File "<pyshell#9>",  line 1, in <module>  xrange(1, 10)  NameError: name 'xrange'  is not defined |
| >>> isinstance(xrange(1, 15), xrange)  True  >>> | >>> isinstance(range(1, 15), range)  True  >>> |

Raising and handling Exceptions

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> 3 < 5.5  True  >>> | >>> 3 < 5.5  True  >>> |
| >>> [3, 5] > 'xyz'  False  >>> | >>> [3, 5] > 'xyz'  Traceback (most recent call last):  File "<pyshell#14>",  line 1, in <module>  [3, 5] > 'xyz'  TypeError: unorderable types:  list() > str()  >>> |
| >>> (3, 5) > 'xyz'  True  >>> | >>> (3, 5) > 'xyz'  Traceback (most recent call last):  File "<pyshell#15>",  line 1, in <module>  (3, 5) > 'xyz'  TypeError: unorderable types:  tuple() > str()>>> |
| >>> [3, 5] > (3, 5)  False  >>> | >>> [3, 5] > (3, 5)  Traceback (most recent call last):  File "<pyshell#21>",  line 1, in <module>  [3, 5] > (3, 5)  TypeError: unorderable types:  list() > tuple()  >>> |
| >>> 50 < [2, 'x'] < 'xyz' < (2, 'x')  True  >>> | >>> 50 < [2, 'x'] < 'xyz' < (2, 'x')  Traceback (most recent call last):  File "<pyshell#23>",  line 1, in <module>  50 < [2, 'x'] < 'xyz' < (2, 'x')  TypeError: unorderable types:  int() < list()  >>> |

Comparison of unorderable types

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> print "[2, 3]>'xyz'= ", [2, 3]>'xyz'  [2, 3]>'xyz' = False  >>> print "(2, 3)>'xyz' = ", (2, 3)>'xyz'  (2, 3)>'xyz' = True  >>> print "[2, 3]>(2, 3) = ", [2, 3]>(2, 3)  [2, 3] > (2, 3) = False | print("[1, 2]>'foo' = ", [1, 2]>'foo')  print("(1, 2)>'foo' = ", (1, 2)>'foo')  print("[1, 2]>(1, 2) = ", [1, 2]>(1, 2))  Traceback (most recent call last):  File "C:/Python34/test.py", line 1,  in  print("[1, 2] > 'foo' = ",  [1, 2] > 'foo')  TypeError: unorderable types:  list() > str() |

bytes vs string

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> a = "Python"  >>> len(a)  6  >>> a[0]  'P'  >>> | >>> a = "Python"  >>> len(a)  6  >>> a[0]  'P'  >>> |
|  | >>>  #To treat a string as a  #sequence of bytes,  #you need to cast:  >>> a = bytes("Python", "utf-8")  >>> a  b'Python'  >>> a[0]  80  >>> |
|  | >>> #bytes(foo, 'utf-8') means to encode <em>foo</em>  >>> #in UTF-8. treat the result as a  >>> #sequence of unsigned 8bit integers. You can  >>> #also convert bytes to a string,  >>> #as following  >>> a = bytes("Python", "utf-8")  >>> b = str(a, "utf-8")  >>> b  'Python'  >>> |

Integer division

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> import sys  >>> print(sys.version)  2.7.9 (default, Apr 2 2015,  15:33:21)[GCC 4.9.2]  >>> print "6/3 =", 6/3  6/3 = 2  >>> print "6//3 =", 6//3  6//3 = 2  >>> print "6/3.0 =", 6/3.0  6/3.0 = 2.0  >>> print "6//3.0 =", 6//3.0  6//3.0 = 2.0  >>> | >>> import sys  >>> print(sys.version)  3.4.2 (v3.4.2:ab2c023a9432,  Oct 6 2014, 22:15:05)  [MSC v.1600 32 bit (Intel)]  >>>  >>> print("6/3 =", 6/3)  6/3 = 2.0  >>> print("6//3 =", 6//3)  6//3 = 2  >>> print("6/3.0 =", 6/3.0)  6/3.0 = 2.0  >>> print("6//3.0 =", 6//3.0)  6//3.0 = 2.0  >>> |

Unicode

Python 2 has ASCII str() types, separate unicode(), but no byte type and in Python 3.0, the language features a [str](https://docs.python.org/3/library/stdtypes.html" \l "str" \t "_blank) type that contain Unicode characters,

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> import sys  >>> print(sys.version)  2.7.9 (default, Apr 2 2015,  15:33:21) GCC 4.9.2]  >>> print type(unicode  ("Python Unicode"))  <type 'unicode'>  >>> | >>> import sys  >>> print(sys.version)  3.4.2 (v3.4.2:ab2c023a9432,  Oct 6 2014, 22:15:05)  MSC v.1600 32 bit (Intel)]  >>> |
| >>> print type(b"Python")  <type 'str'>  >>> |  |
| >>> print "Python" + b" Unicode"  Python Unicode  >>> | >>> print(" Python", type(b" Unicode"))  Python <class 'bytes'>  >>> |

dictionary method

In Python 2   
dict.iteritems(): Return an iterator over the dictionary's (key, value) pairs.

In Python 3   
dict.items(): Return a copy of the dictionary’s list of (key, value) pairs.

|  |  |
| --- | --- |
| **Python 2** | **Python 3** |
| >>> x = {1: "red", 2: "blue"}  >>> print(x)  {1: 'red', 2: 'blue'}  >>> print(x.items())  dict\_items([(1, 'red'), (2, 'blue')])  >>> print(x.iteritems())  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  AttributeError: 'dict' object has no  attribute 'iteritems'  >>> print([i for i in x.iteritems()])  [(1, 'red'), (2, 'blue')]  >>> | >>> x = {1: "red", 2: "blue"}  >>> print(x)  {1: 'red', 2: 'blue'}  >>> print(x.items)  <built-in method items of dict object  at 0x7fe191a89748>  >>> print(x.items())  dict\_items([(1, 'red'), (2, 'blue')])  >>> print([i for i in x.items()])  [(1, 'red'), (2, 'blue')]  >>> |
|  |  |

For other dictionary methods: dict.keys, dict.values and dict.items return a list in Python 2, but in Python 3 they return a view object.

|  |  |
| --- | --- |
| >>> #In Python 2 dict.has\_key()   is used to test.  >>> #whether the dict has  a certain key.  >>> x = {1: "red", 2: "blue"}  >>> print(x.has\_key(1))  True  >>> | >>> #But this method has been  >>> #removed in Python 3.  >>> #Instead, the in operator  >>> #Is used.  >>> x = {1: "red", 2: "blue"}  >>> print(a.has\_key(1))  #Don't run it, unless you want this:  Traceback (most recent call last):  File "<stdin>", line 1, in <module>  AttributeError: 'dict' object has  no attribute 'has\_key'  >>> print(1 in x)  True  >>> |

Data Input

Python 2 :  
raw\_input([prompt]) - If the prompt argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, EOFError is raised.

Python 3 :  
input([prompt]) - If the prompt argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, EOFError is raised.

|  |  |
| --- | --- |
| **Python2** | **Python3** |
| >>>data\_input2 = raw\_input() | >>>data\_input3 = input() |