thon-programming-language-overview

February 1, 2023

[ ]:
```
pwd()
```

[ ]:
```
'C:\\Users\\win10\\jupyter\\Complete Python
Module-20210227T092441Z-001\\Complete Python Module\\1.Python Overview'
```

1 A Recent Study About Programming

1.0.1 Programming 'language': Brain scans reveal coding uses same regions as speech - Article Link

This proves that Programming is nothing but a language that you learn to speak to communicate with the computer. And Python serves best here because of its easy to learn and read syntax.

1.1 1.1 Introduction

- Python is an-
  > * interpreted, > * object-oriented, > * high-level programming language with dynamic semantics.
- It was created by **Guido van Rossum** and first released in the year **1991**.
- It has the following features that makes it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together - > * high-level built in data structures, > * dynamic typing > * dynamic binding
- Python's simple, easy to learn syntax emphasizes on code readability with the use of whitespaces and therefore reduces the cost of program maintenance.
- Python supports modules and packages, which encourages program modularity and code reuse.
- The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.
- Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.

1.2 1.2 Python Version

1.2.1 1.2.1 Python 2

- Published in late 2000, Python 2 signalled a more transparent and inclusive language development process than earlier versions of Python with the implementation of PEP (Python Enhancement Proposal), a technical specification that either provides information to Python community members or describes a new feature of the language.
- Additionally, Python 2 included many more programmatic features including a cycle-detecting garbage collector to automate memory management, increased Unicode support to standardize characters, and list comprehensions to create a list based on existing lists. As Python 2 continued to develop, more features were added, including unifying Python's types and classes into one hierarchy in Python version 2.2.

1.2.2 1.2.2 Python 3

- Python 3 is regarded as the future of Python and is the version of the language that is currently in development. A major overhaul, Python 3 was released in late 2008 to address and amend intrinsic design flaws of previous versions of the language. The focus of Python 3 development was to clean up the codebase and remove redundancy, making it clear that there was only one way to perform a given task.
- Major modifications to Python 3.0 included changing the print statement into a built-in function, improve the way integers are divided, and providing more Unicode support.
- At first, Python 3 was slowly adopted due to the language not being backwards compatible with Python 2, requiring people to make a decision as to which version of the language to use. Additionally, many package libraries were only available for Python 2, but as the development team behind Python 3 has reiterated that there is an end of life for Python 2 support, more libraries have been ported to Python 3. The increased adoption of Python 3 can be shown by the number of Python packages that now provide Python 3 support, which at the time of writing includes 339 of the 360 most popular Python packages.

1.2.3 1.2.3 Python 2.7

- Following the 2008 release of Python 3.0, Python 2.7 was published on July 3, 2010 and planned as the last of the 2.x releases. The intention behind Python 2.7 was to make it easier for Python 2.x users to port features over to Python 3 by providing some measure of compatibility between the two. This compatibility support included enhanced modules for version 2.7 like unittest to support test automation, argparse for parsing command-line options, and more convenient classes in collections.
- Because of Python 2.7's unique position as a version in between the earlier iterations of Python 2 and Python 3.0, it has persisted as a very popular choice for programmers due to its compatibility with many robust libraries. When we talk about Python 2 today, we are typically referring to the Python 2.7 release as that is the most frequently used version.
- Python 2.7, however, is considered to be a legacy language and its continued development, which today mostly consists of bug fixes, will cease completely in 2020.

1.2.4 1.2.4 Key Differences

- While Python 2.7 and Python 3 share many similar capabilities, they should not be thought of as entirely interchangeable. Though you can write good code and useful programs in either version, it is worth understanding that there will be some considerable differences in code syntax and handling.

1.3 1.3 Why to use Python?

Among numerous languages available in the market why should you choose python? This is the first question that arises in the mind of new users.

Following are the som of the reasons why people select python-

- **Quality of software**: Python was meant for readability. Its reusable and maintainable as compared to other languages. Its easier to understand. It supports all the modern features like OOPs and functional programming.
- **Productivity of Developers**: The same program which is written in other high-level languages like c++ or java can be written in one-third or one-fifth line of codes. That means debugging can be easy and it will be less prone to error which in turn increases the productivity of the developers.
- **Portability**: Mostly it's platform-independent. It can run on any platform or OS with minor or no change at all which makes it a highly portable language. Now you can use MircoPython to interact with hardware as well. It can be used on most of the edge devices.
- **Supporting Libraries**: Python already has a lot of inbuilt libraries that come with the standard python package which you download from its official site. With these libraries, you can build lots of basic applications or day to day automation tasks like copying data in bulk from one place to another. Apart from this, there's a huge list of third-party libraries like Numpy, Matplotlib, Scikit Learn, etc.
- **Fun to use**: Its simplicity and availability of lots of supporting libraries plus huge open source community support make development in python a breeze. That's why its widely preferred by hobbyists as well.

1.4 1.4 What can be done with Python?

- **System Programming**
- **Graphical User Interface**
- **Web Scrapping**
- **Managing Database**
- **Fast Prototyping**
- **Numeric / Scientific Programming**
- **Game development**
- **Image Processing**
- **Robotics**
- **Automation**
- **Data science**
- **Data Mining**

1.5 1.5 Zen of Python -

By importing this it displays the basic design philosophies of python

[ ]:
```python
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

day1-data-science-masters

February 1, 2023

```
[16]: a=10
```

```
[17]: a
```

```
[17]: 10
```

```
[20]: ## Comments
      ## for single we use ##
      ## this is a function
      # this is a comment
      a=20
      a
```

```
[20]: 20
```

```
[23]: """
      Example of multiline comments
      hello this is a example of python
      dsfsdfdfdsf
      dfsdfsdfsdfdsf
      dfdsfdsfsdf
      """
      a=20
      a
```

```
[23]: 20
```

1 hello

1.1 hello1

1.1.1 hello2

1. hello guys how are you

1.1.2 Numbers

[26]:
```
1+3
```

[26]:
```
4
```

[27]:
```
print(1+3)
```

```
4
```

[29]:
```
a=5
print(a)
```

```
5
```

[33]:
```
print("hello world my name is krish i work in \n PW skills and ineuron")
```

```
hello world my name is krish i work in
 PW skills and ineuron
```

1.2 Variables Assignment

[34]:
```
name="Krish"
company="PWskills and ineuron"
```

[45]:
```
type("sfdfdfsdf")
```

[45]:
```
str
```

[46]:
```
type(name)
```

[46]:
```
str
```

[ ]:
```

```

[39]:
```
name="Bala"
```

[40]:
```
name
```

[40]:
```
'Bala'
```

[36]:
```
company
```

[36]:
```
'PWskills and ineuron'
```

[37]:
```
number=10
```

[47]:
```
type(number)
```

[47]: int

[41]:
```
number=20
number
```

[41]: 20

[48]:
```
decimal_num=2.5
```

[49]:
```
decimal_num
```

[49]: 2.5

[50]:
```
type(decimal_num)
```

[50]: float

[51]:
```
type("hello")
```

[51]: str

[54]:
```
type(1+2j)
```

[54]: complex

[55]:
```
print('hello')
print("hello")
```

```
hello
hello
```

[ ]:
```
##bad way
## dont start the variable name with numbers
1a
3answer
```

[56]:
```
number1=23
```

[59]:
```
## Variables are case senssitive
company='ineuron'
Company='PWSKILLS'
print(company)
print(Company)
```

```
ineuron
PWSKILLS
```

[ ]:
```
## Reserved keywords
'''
int,float,len,complex,bool,str,return,yield
'''
```

[ ]:
```
float=32
```

[61]:
```
##Boolean
True
```

[61]:
```
True
```

[62]:
```
False
```

[62]:
```
False
```

[63]:
```
True and False
```

[63]:
```
False
```

[64]:
```
True or False
```

[64]:
```
True
```

[67]:
```
not True
```

[67]:
```
False
```

[69]:
```
type(not False)
```

[69]:
```
bool
```

[77]:
```
## Typecasting
bool(0)
```

[77]:
```
False
```

[78]:
```
str(23)
```

[78]:
```
'23'
```

[79]:
```
int('23')
```

[79]:
```
23
```

[80]:
```
type(int('23'))
```

```
[80]: int
```

```
[71]: bool(1)
```

```
[71]: True
```

```
[76]: a=1
      if bool(a)==True:

          print("True")
```

```
True
```

```
[85]: int('123')
```

```
[85]: 123
```

1.2.1 Dynamic Typing

```
[82]: a=12
      str1="Krissh"
      a="var"
```

```
[83]: print(type(a))
```

```
<class 'str'>
```

```
[86]: int(1.54)
```

```
[86]: 1
```

```
[87]: ## concatenation between different types
```

```
[90]: int("1") + "1"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[90], line 1
----> 1 int("1") + "1"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
[92]: "1" * 100
```

```
[92]: '11111111111111111111111111111111111111111111111111111111111111111111111111111111
      11111111111111111111'
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```