PYTHON CORE INTRODUCTION

softserve

More information about Python

http://python.org/

- documentation, tutorials, beginners guide, core distribution, ...

Books include:

Learning Python by Mark Lutz

Python Essential Reference by David Beazley

Python Cookbook, ed. by Martelli, Ravenscroft and Ascher

online at

https://python.swaroopch.com/ (Byte of Python)

https://www.coursera.org/specializations/python

https://www.w3schools.com/python/

https://www.udemy.com/python-101-beginners-coding-bootcamp-free-course/

Python features

Python is an **interpreted**, **interactive**, **object-oriented** programming language.

Python is a **general-purpose**, **high-level** programming language whose design philosophy emphasizes code readability.

Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive.

Its use of indentation for block delimiters is unusual among popular programming languages.

Python features a **dynamic type** system and automatic **memory management**. It supports multiple **programming paradigms**, including **object oriented**, **imperative**, **functional** and **procedural**, and has a large and comprehensive **standard library**.

Simplicity and conciseness

Python:

```
file = open('file.txt')
content = file.read()
```

Java:

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
public class Main {
    public static void main(String[] args) throws IOException {
        String content = new
String(Files.readAllBytes(Paths.get("file.txt")));
    }
}
```

Python history

Python created by Guido van Rossum and first released in 1991.

Python 1.0 - January 1994
- Python 1.5 - December 31, 1997
- Python 1.6 - September 5, 2000

Python 2.0 - October 16, 2000
- Python 2.1 - April 17, 2001
- Python 2.2 - December 21, 2001
- Python 2.3 - July 29, 2003
- Python 2.4 - November 30, 2004
- Python 2.5 - September 19, 2006
- Python 2.6 - October 1, 2008
- Python 2.7 - July 3, 2010

Python 3.0 - December 3, 2008
- Python 3.1 - June 27, 2009
- Python 3.2 - February 20, 2011
- Python 3.3 - September 29, 2012
- Python 3.4 - March 16, 2014
- Python 3.5 - September 13, 2015
- Python 3.6 - December 23, 2016
- Python 3.7 - June 27, 2018
- Python 3.8 - October 14, 2019
- Python 3.9 - October 5, 2020
- Python 3.10 - October 4, 2021

softserve

Python Overview

Cisco, Inc – Web security (e-mail, www)

Yahoo (Maps, Groups)

Google – many components of the Google spider and search engine are written in Python

YouTube – entirely written in Python

Zope Corporation – has developed a powerful Web application framework server using Python

Linux Weekly News

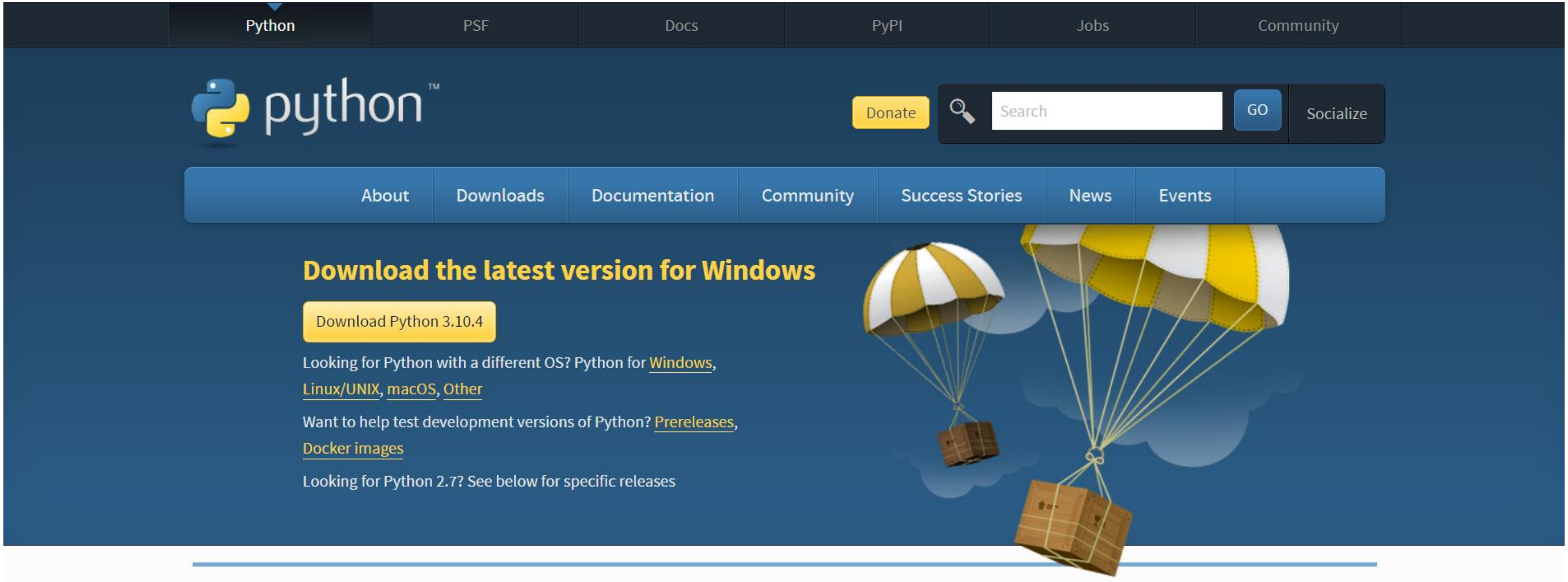
George Lucas' Film Company - **Industry Light and Magic** – uses Python in the production of their FX

Walt Disney Feature Animation

NASA – uses Python in its integrated Planning System and Mission Control Center. Python is going to replace other tools written in Perl and shell dialects

Python download

https://www.python.org/downloads/



Download the latest version for Windows

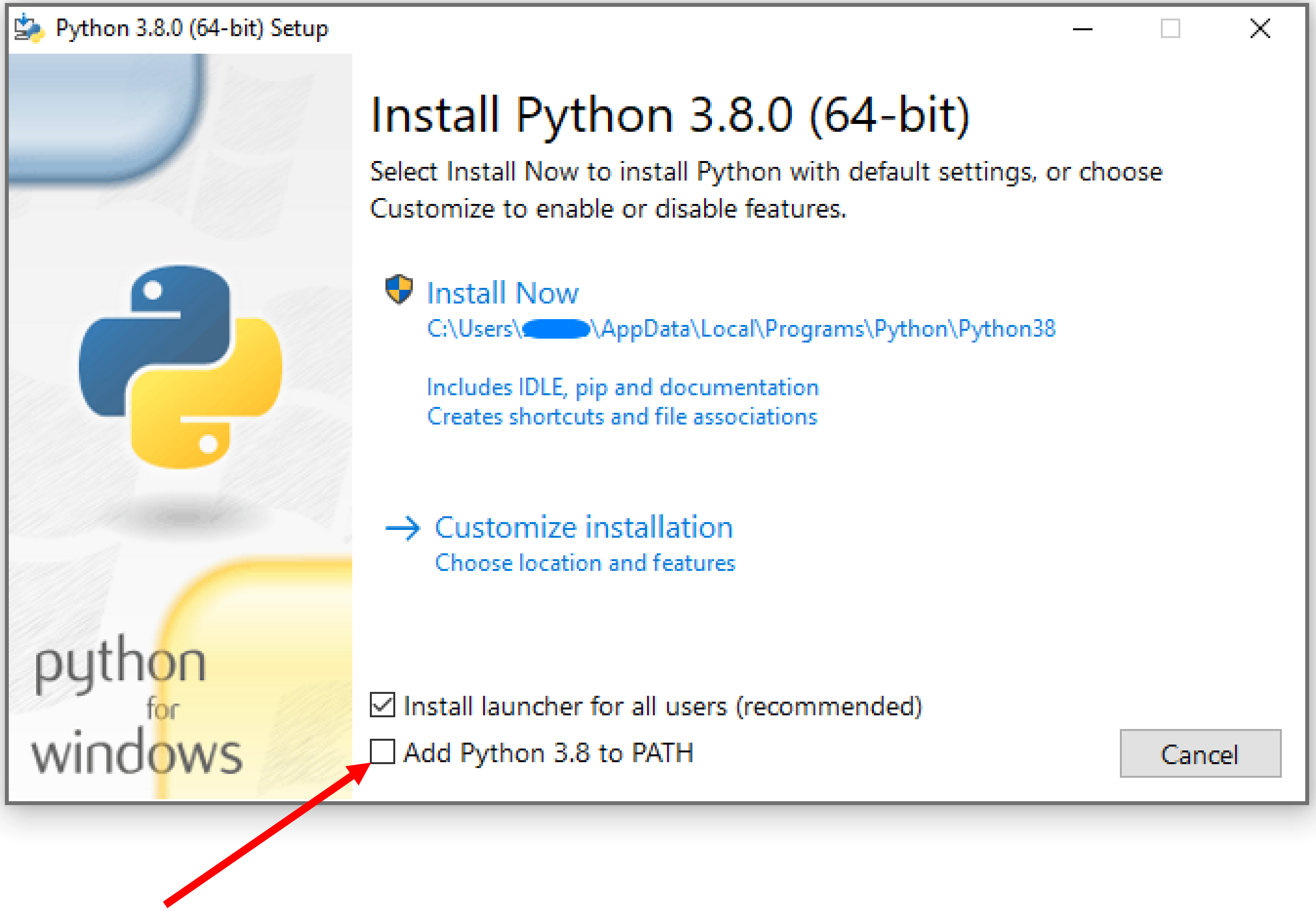
Download Python 3.10.4

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

Python download



Python 3.8.0 (64-bit) Setup

Install Python 3.8.0 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\[redacted]\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

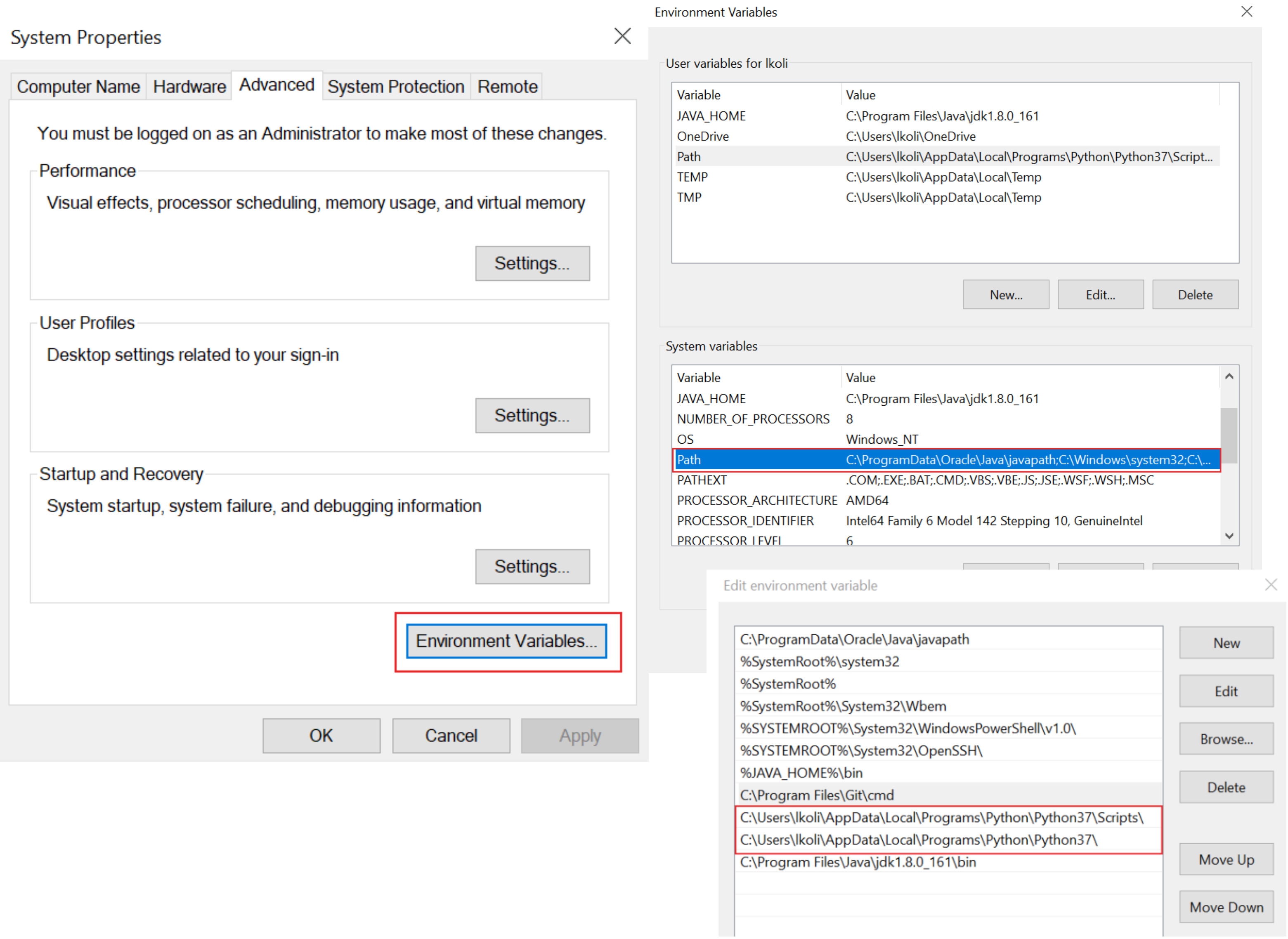
→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)

□ Add Python 3.8 to PATH

Cancel

Install Python



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

User variables for lkoli

Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_161
OneDrive	C:\Users\lkoli\OneDrive
Path	C:\Users\lkoli\AppData\Local\Programs\Python\Python37\Script...
TEMP	C:\Users\lkoli\AppData\Local\Temp
TMP	C:\Users\lkoli\AppData\Local\Temp

System variables

Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_161
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 10, GenuineIntel
PROCESSOR_LEVEL	6

```
C:\ProgramData\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
%JAVA_HOME%\bin
C:\Program Files\Git\cmd
C:\Users\lkoli\AppData\Local\Programs\Python\Python37\Scripts\
C:\Users\lkoli\AppData\Local\Programs\Python\Python37\
C:\Program Files\Java\jdk1.8.0_161\bin
```

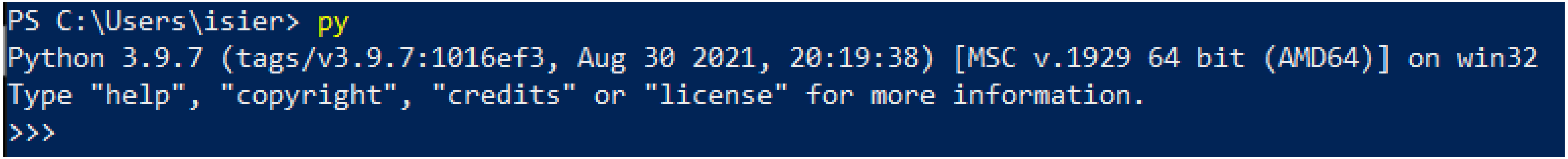
Add Python to the PATH Environmental Variable
You must append your **installation path** (example: C:\Users\*User*\AppData\Local\Programs\Python\Python3*) to the **PATH variable in System**

softserve

Install Python

Next step:

open your command line and type `python` or `py`



```
PS C:\Users\isier> py
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Online Python compiler

- https://www.tutorialspoint.com/execute_python_online.php
- https://www.jdoodle.com/python3-programming-online
- https://www.onlinegdb.com/online_python_compiler
- https://repl.it/repls/DimpledRottenOperation
- https://ideone.com/
- https://www.beta.browxy.com/
- https://paiza.io/projects/sepDWD3s9TLX_8GKIvvbXA?language=python3

Environment, Development Tools

Eclipse + PyDev
Sublime Text
Atom
GNU Emacs
Vi / Vim
Visual Studio
Visual Studio Code
PyCharm by JetBrains
Spyder
Thonny

Python philosophy: The Zen of Python, by Tim Peters

import this

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!

Python syntax

softserve

Python Syntax

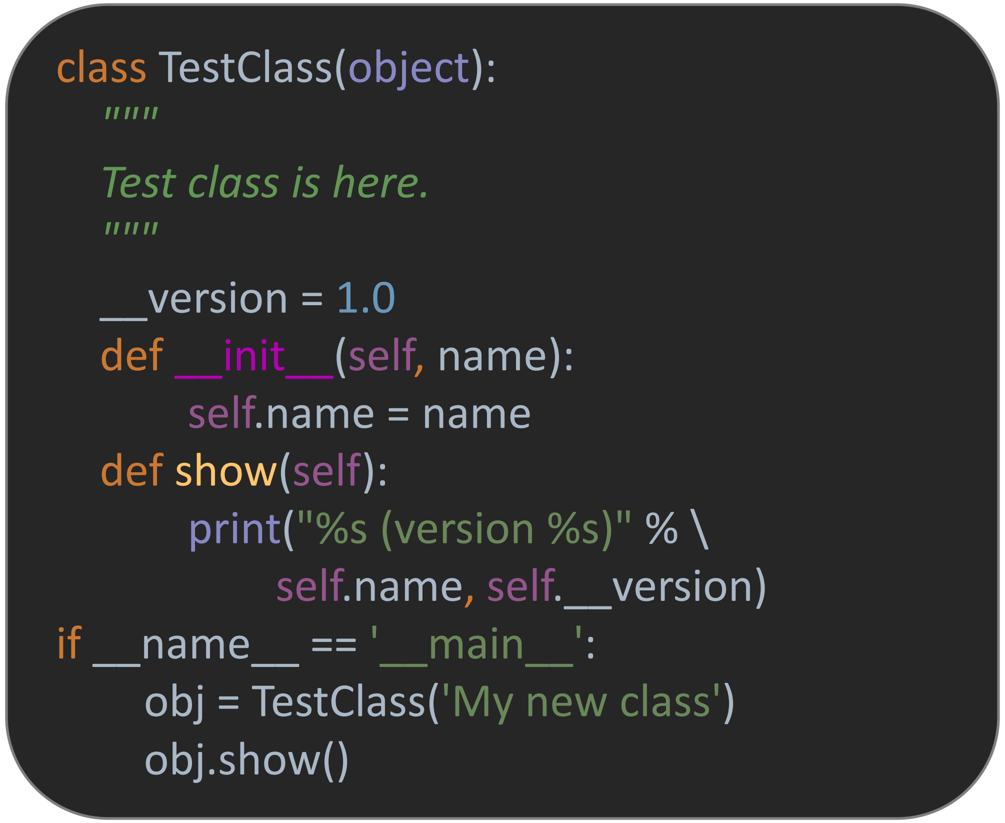
Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, frequently using English keywords where other languages use punctuation. Python requires less boilerplate than traditional manifestly typed structured languages such as C or Pascal, and has a smaller number of syntactic exceptions and special cases than either of these

Indentation - Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks (a feature also known as the off-side rule):

increase in indentation comes after certain statements;

decrease in indentation signifies the end of the current block

Use '\' when must go to next line prematurely



```
class TestClass(object):
    """

    Test class is here.
    """

    __version = 1.0
    def __init__(self, name):
        self.name = name
    def show(self):
        print("%s (version %s)" % \
            self.name, self.__version)
if __name__ == '__main__':
    obj = TestClass('My new class')
    obj.show()
```

Python Variables

In most of the programming languages a **variable** is a named location used to store data in the memory. Each variable must have a unique name called identifier. It is helpful to think of variables as container that hold data which can be changed later throughout programming.

In Python we don't assign values to the variables, where as Python gives the **reference** of the object (value) to the variable.

In Python, variables **do not need declaration** to reserve memory space. The **"variable declaration"** or **"variable initialization"** happens **automatically** when we assign a value to a variable.

A process in which a variable is set to its first value is called ***initialization***.

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

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.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

Keywords cannot be used as identifiers.

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

Python Identifiers

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.
- We can also use different styles of writing, i.e., capitalize every first letter of the word except the initial word without any spaces. For example: camelCase, PascalCase, snake_case, kebeb-case

Python Syntax - Enough to Understand the Code

- Assignment uses = and comparison uses ==.
- For numbers +-*/% are as expected.
- Special use of + for string concatenation.
- Special use of % for string formatting.
- Logical operators are words (**and**, **or**, **not**) not symbols (&&, ||, !).
- First assignment to a variable will create it.
- Variable types don’t need to be declared.
- Python figures out the variable types on its own.

Variables

- No need to declare
- The variable name is case sensitive: ‘val’ is not the same as ‘Val’
- Variables are created when they are assigned
- A variable can be reassigned to whatever, whenever (functions, modules, classes)
- The type of the variable is determined by Python

Modules and packages

softserve

Modules

- A module is a file containing Python definitions and statements
- File should have *suffix*.py
- Within a module, the module’s name is available as through global variable _name_.
- Use “import module-name” to import the functions in this module
- It is not required to place all import statements at the beginning of a module
- Some modules are built-in e.g. sys

Modules

Import module fibo

```
>>> import fibo
```

Using the module name you can access the functions

```
>>> fibo.fib(1000)
 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
 [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__ 'fibo'
```

```
>>> fib = fibo.fib
>>> fib(500) 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import fib, fib2
>>> fib(500) 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
>>> from fibo import *
>>> fib(500)
```

```
# Fibonacci numbers module fibo.py

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

softserve

Python Package

- Packages are a way of structuring Python's module namespace by using "dotted module names"
- When importing the package, Python searches through the directories on sys.path looking for the package subdirectory.
- To import module C you can use:

  import A.B.C and use the fully qualified name **print(A.B.C.my_func())**

- To import module or function from the package use following:

  from A.B import C and use C only by its name (without package prefix) **print (C.my_func())**

- The __init__.py files are required for Python package(can be empty but must be present in package dir)
- Location of Python package can be localpath or must be defined in "sys.path" (PATH):

```
import sys
sys.path
sys.path.append(<location>)
```

Using packages in Python

```
fincalc
            |-- __init__.py
            |-- simper.py
            |-- compper.py
            |-- annuity.py
```

Module __init.py__ may be empty, or may contain a variable __all__

```
__all__ = ["simper", "compper", "annuity"]
```

Structure of packages

```
Sound                               # main packege
           __init__.py              #nitialization of package

           Formats                  # sub packege
                      __init__.py
                      wavread.py
                      wavwrite.py
                      aiffread.py
                      aiffwrite.py
                      ...
           Effects                  #  sub packege
                       __init__.py
                      echo.py
                      surround.py
                      reverse.py
           Filters                  #  sub packege
                      __init__.py
                      equalizer.py
                      vocoder.py
```

```
>>> import Sound.Effects.echo

>>>Sound.Effects.echo.echofilter(delay=0.7, atten=4)

>>>from Sound.Effects import echo

>>>echo.echofilter(delay=0.7, atten=4)

>>>from Sound.Effects.echo import echofilter

>>>echofilter(input, output, delay=0.7, atten=4)

>>>from Sound.Effects.echo import *

>>>echofilter(input, output, delay=0.7, atten=4)
```

What is a pip

softserve

What is a pip

pip is a package management system used to install and manage software packages written in Python.

PyPI - https://pypi.python.org/pypi (which you'll occasionally see referred to as The Cheeseshop) is a repository for open-source third-party Python packages. It's similar to RubyGems in the Ruby world, PHP's Packagist, CPAN for Perl, and NPM for Node.js.

The most common scenario is to install from PyPI using Requirement Specifiers.

Python actually has another, more primitive, package manager called easy_install, which is installed automatically when you install Python itself. pip is vastly superior to easy_install for lots of reasons, and so should generally be used instead.

Installation

Per http://www.pip-installer.org/en/latest/installing.html:

Download get-pip.py, being careful to save it as a .py file rather than .txt. Then, run it from the command prompt:

```
>>>python get-pip.py
```

You possibly need an administrator command prompt to do this.

pip is already installed if you're using Python 2 >=2.7.9 or Python 3 >=3.4 binaries downloaded from python.org, but you'll need to upgrade pip.

Installing Packages

pip supports installing from PyPI, version control, local projects, and directly from distribution files.

The most common scenario is to install from PyPI using Requirement Specifiers

```
$ pip install SomePackage                  # latest version
$ pip install SomePackage  ==1.0.4         # specific version
$ pip install SomePackage  >=1.3           # minimum version
$ pip install SomePackage  !=3.5           # Version Exclusion. Anything except version 3.5
$ pip install SomePackage  ~=2.1           # Compatible release. Same as >= 2.1, == 2.*
```

Requirements files

Logically, a Requirements file is just a list of pip install arguments placed in a file. Note that you should not rely on the items in the file being installed by pip in any particular order.

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

```
###### Requirements without Version Specifiers ######
nose
nose-cov
beautifulsoup4
###### Requirements with Version Specifiers ######
docopt == 0.6.1          # Version Matching. Must be version 0.6.1
keyring >= 4.1.1         # Minimum version 4.1.1
coverage != 3.5          # Version Exclusion. Anything except version 3.5
Mopidy-Dirble ~= 1.1     # Compatible release. Same as >= 1.1, == 1.*
###### Refer to other requirements files ######
-r other-requirements.txt
###### A particular file ######
./downloads/numpy-1.9.2-cp34-none-win32.whl
http://wxpython.org/Phoenix/snapshot-builds/wxPython_Phoenix-3.0.3.dev1820+49a8884-cp34-none-win_amd64.whl
###### Additional Requirements without Version Specifiers ######
#   Same as 1st section, just here to show that you can put things in any order.
rejected
green
```

Pip commands

install	Install packages.
download	Download packages.
uninstall	Uninstall packages.
freeze	Output installed packages in requirements format.
list	List installed packages.
show	Show information about installed packages.
search	Search PyPI for packages.
wheel	Build wheels from your requirements.
hash	Compute hashes of package archives.
completion	A helper command used for command completion
help	Show help for commands.

Virtual environments

softserve

Understanding Virtual Environments

“**virtualenv**” creates an isolated Python environment in a directory structure which contains the “site-packages” directory. When we activate the virtual environment and install packages, the packages are placed in the virtual environment's “site-packages” directory instead of Python Installation's site-packages directory.

Install virtualenv using pip.

pip install virtualenv

Usage

Virtualenv has one basic command:

```
$ virtualenv ENV
```

Where **ENV** is a directory to place the new virtual environment. It has a number of usual effects (modifiable by many Options):

- **ENV/lib/** and **ENV/include/** are created, containing supporting library files for a new virtualenv python. Packages installed in this environment will live under **ENV/lib/pythonX.X/site-packages/**.
- **ENV/bin** is created, where executables live - noticeably a new python. Thus running a script with #! **/path/to/ENV/bin/python** would run that script under this virtualenv's python.
- The crucial packages pip and setuptools are installed, which allow other packages to be easily installed to the environment. This associated pip can be run from **ENV/bin/pip**.

The python in your new virtualenv is effectively isolated from the python that was used to create it.

Activate/deactivate

In a newly created virtualenv there will also be a activate shell script. For Windows systems, activation scripts are provided for the Command Prompt and Powershell.

On Posix systems, this resides in /ENV/bin/, so you can run:

```
$ source bin/activate
```

The **activate** script will also modify your shell prompt to indicate which environment is currently active. To disable this behaviour, see *VIRTUAL_ENV_DISABLE_PROMPT*.

To undo these changes to your path (and prompt), just run:

```
$ deactivate
```

On Windows, the equivalent activate script is in the Scripts folder.

.\EVN\scripts\activate

TO
THE
FUTURE
softserve