Course Era

* <https://www.python.org/shell/>
* <https://www.onlinegdb.com/online_python_interpreter>
* <https://repl.it/languages/python3>
* <https://www.tutorialspoint.com/execute_python3_online.php>
* <https://rextester.com/l/python3_online_compiler>
* <https://trinket.io/python3>
* Read the [official Python documentation](https://docs.python.org/3/).
* Search for answers or ask a question on [Stack Overflow](https://stackoverflow.com/).
* Subscribe to the Python [tutor](https://mail.python.org/mailman/listinfo/tutor) mailing list, where you can ask questions and collaborate with other Python learners.
* Subscribe to the [Python-announce](https://mail.python.org/mailman/listinfo/python-announce-list) mailing list to read about the latest updates in the language.

**Python history and current status**

Python was released almost 30 years ago and has a rich history. You can read more about it on the [History of Python](https://en.wikipedia.org/wiki/History_of_Python) Wikipedia page or in the section on the [history of the software](https://docs.python.org/3.0/license.html) from the official Python documentation.

Python has recently been called the fastest growing programming language. If you're interested in why this is and how it’s measured, you can find out more in these articles:

* [The Incredible Growth of Python](https://stackoverflow.blog/2017/09/06/incredible-growth-python/) (Stack Overflow)
* [Why is Python Growing So Quickly - Future Trends](https://www.netguru.com/blog/why-python-is-growing-so-quickly-future-trends) (Netguru)
* [By the numbers: Python community trends in 2017/2018](https://opensource.com/article/18/5/numbers-python-community-trends) (Opensource.com)
* [Developer Survey Results 2018](https://insights.stackoverflow.com/survey/2018#technology) (Stack Overflow)

Python Interpreter codepads

Check python key words

Golden ration in python

* [Wikipedia Recursion page](https://en.wikipedia.org/wiki/Recursion)
* See what happens when you [Search Google for Recursion](https://www.google.com/search?q=recursion)

<https://docs.python.org/3/library/stdtypes.html#string-methods>

<https://www.coursera.org/learn/python-crash-course/supplement/JbXSA/formatting-strings-cheat-sheet>

<https://www.coursera.org/learn/python-crash-course/supplement/sbRdF/lists-and-tuples-operations-cheat-sheet>

<https://www.coursera.org/learn/python-crash-course/supplement/Cc19J/dictionary-methods-cheat-sheet>

* [Installing Python 3 on Windows 10 with Chocolatey](https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-windows-10)
* [Installing Python 3 on MacOS with Homebrew](http://www.pyladies.com/blog/Get-Your-Mac-Ready-for-Python-Programming/)
* [Package management basics on Linux](https://www.digitalocean.com/community/tutorials/package-management-basics-apt-yum-dnf-pkg)

These are some of the common editors for Python, available for all platforms:

* [Atom](https://atom.io/)
* [Eclipse](http://www.eclipse.org/)
* [PyCharm](https://www.jetbrains.com/pycharm/)
* [Sublime Text](http://www.sublimetext.com/)
* [Visual Studio Code](https://code.visualstudio.com/)

You can read more about these editors, and others, in these overview comparatives:

* [Python IDEs and Code Editors (Guide)](https://realpython.com/python-ides-code-editors-guide/#pycharm)
* [Best Python IDEs and Code Editors](https://www.softwaretestinghelp.com/python-ide-code-editors/)
* [Top 5 Python IDEs for Data Science](https://www.datacamp.com/community/tutorials/data-science-python-ide)

<https://xkcd.com/1205/>

<https://www.techrepublic.com/article/python-where-to-learn-it-and-why-you-should-do-it-now/>

<https://online-learning.harvard.edu/course/cs50-introduction-computer-science>

<https://docs.python.org/3/library/functions.html#open>

* <https://docs.python.org/3/library/os.html>
* <https://docs.python.org/3/library/os.path.html>
* <https://en.wikipedia.org/wiki/Unix_time>
* <https://docs.python.org/3/library/csv.html>
* <https://realpython.com/python-csv/>
* <https://docs.python.org/3/howto/regex.html>
* <https://docs.python.org/3/library/re.html>
* <https://docs.python.org/3/howto/regex.html#greedy-versus-non-greedy>

Shout out to [regex101.com](http://regex101.com/), which will explain each stage of a regex.

<https://regexcrossword.com/>

<https://www.regular-expressions.info/python.html>

* <https://docs.python.org/2/library/functions.html#input>
* <https://docs.python.org/2/library/functions.html#raw_input>
* <https://docs.python.org/2/library/functions.html#eval>
* <https://docs.python.org/3/library/functions.html#input>
* [https://docs.python.org/3/library/functions.html#](https://docs.python.org/3/library/functions.html#input)eval

<https://docs.python.org/3/library/subprocess.html>

<https://www.coursera.org/lecture/agile-atlassian-jira/jira-overview-IA6Dg>

**Best of Unit Testing Standard Library Module**

Understand a Basic Example:

* <https://docs.python.org/3/library/unittest.html#basic-example>

Understand how to run the tests using the Command Line:

* <https://docs.python.org/3/library/unittest.html#command-line-interface>

Understand various Unit Test Design Patterns:

* <https://docs.python.org/3/library/unittest.html#organizing-test-code>
* Understand the uses of setUp, tearDown; setUpModule and tearDownModule

Understand basic assertions:

|  |  |  |
| --- | --- | --- |
| **Method** | **Checks that** | **New in** |
| [assertEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEqual) | a == b |  |
| [assertNotEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotEqual) | a != b |  |
| [assertTrue(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTrue) | bool(x) is True |  |
| [assertFalse(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertFalse) | bool(x) is False |  |
| [assertIs(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIs) | a is b | 3.1 |
| [assertIsNot(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNot) | a is not b | 3.1 |
| [assertIsNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNone) | x is None | 3.1 |
| [assertIsNotNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNotNone) | x is not None | 3.1 |
| [assertIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIn) | a in b | 3.1 |
| [assertNotIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIn) | a not in b | 3.1 |
| [assertIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsInstance) | isinstance(a, b) | 3.2 |
| [assertNotIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIsInstance) | not isinstance(a, b) | 3.2 |

Understand more specific assertions such as assertRaises

* <https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaises>
* [Jupyter Notebook Tutorial](https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook), by datacamp.com
* [How to use Jupyter Notebooks](https://www.codecademy.com/articles/how-to-use-jupyter-notebooks), by codeacademy.com
* [Teaching and Learning with Jupyter](https://jupyter4edu.github.io/jupyter-edu-book/), by university professors using Jupyter
* <https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/>
* <https://landing.google.com/sre/sre-book/chapters/testing-reliability/>
* <https://landing.google.com/sre/sre-book/chapters/testing-reliability/>
* <https://testing.googleblog.com/2007/10/performance-testing.html>
* <https://www.guru99.com/smoke-testing.html>
* <https://www.guru99.com/exploratory-testing.html>
* <https://testing.googleblog.com/2008/09/test-first-is-fun_08.html>

Raise allows you to throw an exception at any time.

* <https://docs.python.org/3/tutorial/errors.html#raising-exceptions>

Assert enables you to verify if a certain condition is met and throw an exception if it isn’t.

* [https://docs.python.org/3/reference/simple\_stmts.html#the-assert-statement](https://docs.python.org/2/reference/simple_stmts.html#the-assert-statement)<https://stackoverflow.com/questions/5142418/what-is-the-use-of-assert-in-python>

The standard library documentation is kind of unclear. Basically `assert <something false>` will raise AssertionError, which the caller may need to handle.

In the try clause, all statements are executed until an exception is encountered.

* [https://docs.python.org/3/tutorial/errors.html#handling-exceptions](https://docs.python.org/2/tutorial/errors.html#handling-exceptions)

Except is used to catch and handle the exception(s) that are encountered in the try clause.

* <https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

Other interesting Exception handling readings:

* <https://doughellmann.com/blog/2009/06/19/python-exception-handling-techniques/>

Cisco Live

https://github.com/clay584/cleur\_genie\_demo

<https://www.coursera.org/learn/python-operating-system/supplement/bC5zL/basic-linux-commands-cheat-sheet>

**Managing streams**

These are the redirectors that we can use to take control of the streams of our programs

* command**>** file: redirects standard output, overwrites file
* command **>>** file: redirects standard output, appends to file
* command**<** file: redirects standard input from file
* command **2>** file: redirects standard error to file
* command1 **|** command2: connects the output of command1 to the input of command2

**Operating with processes**

These are some commands that are useful to know in Linux when interacting with processes. Not all of them are explained in videos, so feel free to investigate them on your own.

* **ps:** lists the processes executing in the current terminal for the current user
* **ps**ax: lists all processes currently executing for all users
* **ps** e: shows the environment for the processes listed
* **kill** PID: sends the SIGINT signal to the process identified by PID
* **fg**: causes a job that was stopped or in the background to return to the foreground
* **bg:** causes a job that was stopped to go to the background
* **jobs:** lists the jobs currently running or stopped
* **top:** shows the processes currently using the most CPU time (press "q" to quit)
* <https://ryanstutorials.net/bash-scripting-tutorial/>
* <https://linuxconfig.org/bash-scripting-tutorial-for-beginners>
* <https://www.shellscript.sh>

On top of search results, here are some great Git resources available online:

* [Pro Git](https://git-scm.com/book/en/v2): This book (available online and in print) covers all the fundamentals of how Git works and how to use it. Refer to it if you want to learn more about the subjects that we cover throughout the course.
* [Git tutorial](https://git-scm.com/docs/gittutorial): This tutorial includes a very brief reference of all Git commands available. You can use it to quickly review the commands that you need to use.