Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

• Python is open-source and has a great support community,

• Plus, extensive support libraries.

• Its data structures are user-friendly.

Table of Contents

* Main Python Data Types
* How to Create a String in Python
* Math Operators
* How to Store Strings in Variables
* Built-in Functions in Python
* How to Define a Function
* List
* List Comprehensions
* Tuples
* Dictionaries
* If Statements (Conditional Statements) in Python
* Python Loops
* Class
* Dealing with Python Exceptions (Errors)

Python Basics: Getting Started

Most Windows and Mac computers come with Python pre-installed.

You can check that via a Command Line search.

The particular appeal of Python is that you can write a program in any text editor, save it in .py format and then run via a Command Line.

But as you learn to write more complex code or venture into data science, you might want to switch to an IDE or IDLE.

What is IDLE (Integrated Development and Learning) IDLE (Integrated Development and Learning Environment) comes with every Python installation.

Its advantage over other text editors is that it highlights important keywords (e.g. string functions), making it easier for you to interpret code.

Shell is the default mode of operation for Python IDLE. In essence, it’s a simple loop that performs that following four steps:

• Reads the Python statement

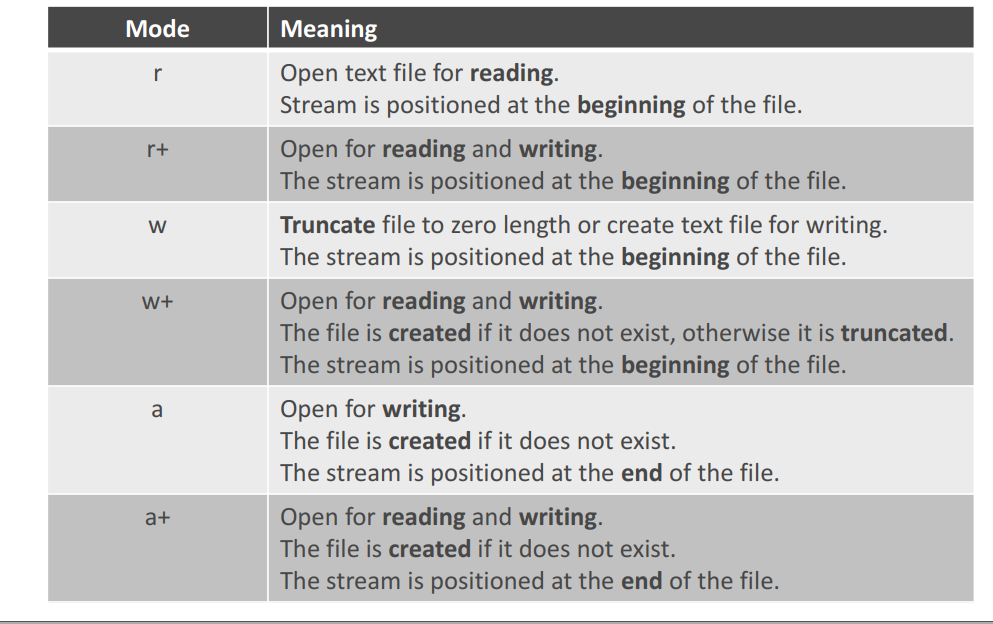
• Evaluates the results of it

• Prints the result on the screen

• And then loops back to read the next statement.

Python shell is a great place to test various small code snippets.

* Opening file modes:



Good

1. Tuples. A tuple is simply a grouping of elements that can be of the same or different types For example, a 3-element tuple describing a book could be (title, author, number\_of\_pages).
2. Returning multiple values. In Python, a function or a method can return multiple values easily be returning a tuple. Combining this with tuple unpacking, you can write code like this:

max, min, avg = summarize(source)

This is also very natural and useful. In Java, I sometimes created simple result classes and returned those when I needed to return multiple values. But it took some effort, which meant that I didn’t use it as much as I would have liked.

1. Functions as arguments. In Python, functions and methods are first class objects. This means for example that they can be passed in as arguments to other functions and methods. In my opinion, this is much cleaner and easier than the Java way with interfaces (often combined with ugly anonymous implementations).
2. Stand-alone functions. In Python, you can define functions that stand by themselves, unlike in Java, where they always have to belong to a class. In Java I frequently used static methods for “functional” tasks (functions that didn’t require any state, but instead only depended on the arguments passed in).
3. Odds and ends. There are a few other features that were pleasant surprises, for example named arguments and default arguments, the built-in function enumerates, flexible string quoting (double quote or single quote, which makes it very easy to write a string containing quotation marks). I also found out that you can use the import-statement within a function, to make the dependency local to the function.

Bad

1. No Static Types. The biggest drawback for me is that there is no type information when reading code. In Java, when I saw a new function and tried to understand what it did, I almost always looked at the types of the arguments and return value to get a sense of what it did. In Python, it is much harder. It takes a lot more digging to find out what a function does in Python.
2. No Static Types. The other consequence of the dynamic typing in Python is that the “find usage” feature in the IDE is not as useful. I am using PyCharm for Python, and used IntelliJ IDEA for Java. In Java, I used “find usage” a lot to find out where a method is called from, where a class is used etc. In PyCharm the accuracy is (naturally) worse – often there are many false positives for usage, unless the name of the function is unique. So it is basically back to grep:ing the code base for a given string (like in my C++ days).

|  |  |
| --- | --- |
| Level: | Beginner – Python enables a beginner to become productive quickly |
| Skills Needed: | Problem-solving, abstract thinking |
| Platform: | Web, Desktop |
| Popularity Among Programmers: | Becoming continuously more popular |
| Benefits: | * Flexible * Naturally/Intuitively readable * Highly regarded official tutorials and documentation * Scripted as opposed to compiled |
| Downsides: | Doesn’t start with programming basics (known to abstract too many important basic concepts) |
| Popularity: | Becoming continuously more popular both in technical education and business uses |
| Degree of Use: | Coding skills widely used; popular in both technical education and business use |
| Annual Salary Projection: | $72,500 |

Overloaden operators

|  |  |
| --- | --- |
| **Math** | **operators** |
| **+** | \_\_add\_\_(self, other) |
| **–** | \_\_sub\_\_(self, other) |
| **\*** | \_\_mul\_\_(self, other) |
| **/** | \_\_truediv\_\_(self, other) |
| **//** | \_\_floordiv\_\_(self, other) |
| **%** | \_\_mod\_\_(self, other) |
| **\*\*** | \_\_pow\_\_(self, other) |
| >> | \_\_rshift\_\_(self, other) |
| << | \_\_lshift\_\_(self, other) |
| & | \_\_and\_\_(self, other) |
| | | \_\_or\_\_(self, other) |
| ^ | \_\_xor\_\_(self, other) |

**Comparison Operators:**

|  |  |
| --- | --- |
| Operator | Magic Method |
| **<** | \_\_lt\_\_(self, other) |
| **>** | \_\_dt\_\_(self,other) |
| **<=** | \_\_le\_\_(SELF, OTHER) |
| **>=** | \_\_ge\_\_(SELF, OTHER) |
| **==** | \_\_eq\_\_(SELF, OTHER) |
| **!=** | \_\_ne\_\_(SELF, OTHER) |

**Assignment Operators:**

|  |  |
| --- | --- |
| Operator | Magic Method |
| **-=** | \_\_isub\_\_(SELF, OTHER) |
| **+=** | \_\_iadd\_\_(SELF, OTHER) |
| **\*=** | \_\_imul\_\_(SELF, OTHER) |
| **/=** | \_\_idiv\_\_(SELF, OTHER) |
| **//=** | \_\_ifloordiv\_\_(SELF, OTHER) |
| **%=** | \_\_imod\_\_(SELF, OTHER) |
| **\*\*=** | \_\_ipow\_\_(SELF, OTHER) |
| **>>=** | \_\_irshift\_\_(SELF, OTHER) |
| **<<=** | \_\_ilshift\_\_(SELF, OTHER) |
| **&=** | \_\_iand\_\_(SELF, OTHER) |
| **|=** | \_\_ior\_\_(SELF, OTHER) |
| **^=** | \_\_ixor\_\_(SELF, OTHER) |

**Unary Operators:**

|  |  |
| --- | --- |
| Operator | Magic Method |
| **–** | \_\_neg\_\_(SELF, OTHER) |
| **+** | \_\_pos\_\_(SELF, OTHER) |
| **~** | \_\_invert\_\_(SELF, OTHER) |

Tkinter events

<Button-1> Button 1 is the leftmost button, button 2 is the middle button

(where available), and button 3 the rightmost button.

<Button-1>, <ButtonPress-1>, and <1> are all synonyms.

For mouse wheel support under Linux, use Button-4 (scroll

up) and Button-5 (scroll down)

<B1-Motion> The mouse is moved, with mouse button 1 being held down (use

B2 for the middle button, B3 for the right button).

<ButtonRelease-1> Button 1 was released. This is probably a better choice in

most cases than the Button event, because if the user

accidentally presses the button, they can move the mouse

off the widget to avoid setting off the event.

<Double-Button-1> Button 1 was double clicked. You can use Double or Triple as

prefixes.

<Enter> The mouse pointer entered the widget (this event doesn’t mean

that the user pressed the Enter key!).

<Leave> The mouse pointer left the widget.

<FocusIn> Keyboard focus was moved to this widget, or to a child of

this widget.

<FocusOut> Keyboard focus was moved from this widget to another widget.

<Return> The user pressed the Enter key. For an ordinary 102-key

PC-style keyboard, the special keys are Cancel (the Break

key), BackSpace, Tab, Return(the Enter key), Shift\_L (any

Shift key), Control\_L (any Control key), Alt\_L (any Alt key),

Pause, Caps\_Lock, Escape, Prior (Page Up), Next (Page Down),

End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1,

F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num\_Lock, and

Scroll\_Lock.

<Key> The user pressed any key. The key is provided in the char

member of the event object passed to the callback (this is an

empty string for special keys).

a The user typed an “a”. Most printable characters can be used

as is. The exceptions are space (<space>) and less than

(<less>). Note that 1 is a keyboard binding, while <1> is a

button binding.

<Shift-Up> The user pressed the Up arrow, while holding the Shift key

pressed. You can use prefixes like Alt, Shift, and Control.

<Configure> The widget changed size (or location, on some platforms). The

new size is provided in the width and height attributes of

the event object passed to the callback.

<Activate> A widget is changing from being inactive to being active.

This refers to changes in the state option of a widget such

as a button changing from inactive (grayed out) to active.

<Deactivate> A widget is changing from being active to being inactive.

This refers to changes in the state option of a widget such

as a radiobutton changing from active to inactive (grayed out).

<Destroy> A widget is being destroyed.

<Expose> This event occurs whenever at least some part of your

application or widget becomes visible after having been

covered up by another window.

<KeyRelease> The user let up on a key.

<Map> A widget is being mapped, that is, made visible in the

application. This will happen, for example, when you call the

widget's .grid() method.

<Motion> The user moved the mouse pointer entirely within a widget.

<MouseWheel> The user moved the mouse wheel up or down. At present, this

binding works on Windows and MacOS, but not under Linux.

<Unmap> A widget is being unmapped and is no longer visible.

<Visibility> Happens when at least some part of the application window

becomes visible on the screen.