# Python

**Pip**

While working as a developer, you’ll find yourself wanting to use some functionality that is not included in your language’s standard library. For example, in a standard .py file or document you will only be able to work with the standard library tools packaged with the language.

However, as a language grows and is used for more purposes, developers build their own tools and package them for use by others. These are known as third party modules. Pip is the package manager we use in Python to install these modules. Installing third party modules is pretty simple, because pip does a great job at knowing which version of that package we need to match our Python version.

**Comments**

Comments are useful because they allow you to explain what your code is doing. Every language has a way of ensuring that some lines will not be executed at run time.

As a developer, one of the most important jobs is writing re-usable code. By explaining what our code does in comments, we make it easier for ourselves and others to edit our code later on. In addition, comments can be helpful for writing pseudo-code when you’re trying to work out a tough problem.

# commenting a single line

# we can even comment out code

# print "this will not print!"

print "read below for more on multi-line comments in python!"

copy

#this would execute

# This line and below would not execute

'''

Triple quotations allow us to comment across multiple lines as long as

the triple quoted comment is not the first thing in your file.

You can use double or single quotes!

'''

**Data Types**

Data type refers to how the computer knows to classify information. To determine data type, ask what category a value belongs to. Here’s a list of the data types that you will surely be using in building web applications.

There are several general classifications for data we’re interested in. **Primitive data types** are the basic building blocks of a language. Most languages have these in common. Here are the most common:

* **Boolean-** Assesses the truth value of something. It has only two values: True & False
* **Numbers-** Integers (whole numbers), floating point numbers (commonly known as decimal numbers), and complex numbers.
* **strings-** A text literal. Most pages in the web work with strings quite often.

**Composite types** are collections composted of the above primitive types**.**

* **Tuples-** A type of data that is immutable (can’t be modified after its creation) and can hold a group of values. **Tuples** can contain mixed data types.
* **Lists-** A type of data that is mutable and can hold a group of values. Usually meant to store a collection of related data.
* **Dictionaries-** A group of key-value pairs. Dictionary elements are indexed by unique keys which are used to access values.

## In Python, (almost) everything is an object. We will touch on this later when we get into Object Oriented Programming(OOP).

**Indentation & Line-Endings**

One of the most important aspects of Python is indentation. Python has no brackets, braces, or keywords to indicate the start and code itself. You’ll see that indenting starts a new code block and un-indenting ends that block. Don’t worry if these codes don’t make sense right now; we’ll go over function and if- statements later. Just take not of how the indentation looks.

Conditional Expression

age = 16

if age >= 18:

print "legal age"

elif age == 17 or age < 17:

print "you are so young!"

my\_list = [4, "dog", ['german', 'shepherd'], 'bitme']

for element in my\_list:

print element

for count in range(0,100):

print "looping -", count

count = 0

while count < 5:

print "looping -", count

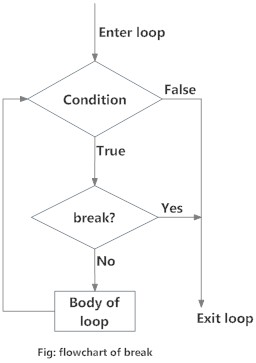
count += 1

x = [19,2,54,-2,7,12,98,32,10,-3,6]

x.sort

y,z=[i for i in x if i<0 ],[j for j in x if j>0]

print y,z //splits list at 0



**Functions**

A function is a named block of code that we can execute to perform a specific task. More simply, a function is a list of instructions that you can run at any time. If you find something that you seem to be using over and over again, it might be best to have a way to streamline the process. A function can optionally take in parameters, perform a series of instructions, and optionally return something afterwards. Here’s an example:

def add(a,b)

x = a + b

return x

result = add(3,5)

print result

Think of the function as a factory. If we were building a new car we would:

* Acquire raw materials (variables) needed for creating a car.
* Send the raw materials (invoke and pass arguments) to a car manufacturing plant (function)
* Do something (process) with the raw materials(parameters)
* Drive the car (function’s return value)

The factory has all the instructions to build a new car and will perform all the tasks. When you want a new car, all you have to do is call the factory to request a new car.

The advantages of using functions are:

* Reducing the duplication of code
* Breaking down complex problems into simpler pieces
* Improving clarity of code

**Syntax**

Pay attention to a few details. The **def** keyword signifies the declaration of a function. This indicates that **the following code is a function and assigns a name to that function, so we can call it later**. Parameters are information we input into a function, and appear inside the parenthesis that follow the function name.

**Function Parameters**

We define the input of functions using parameters. Like we’ve seen before, some functions do not have to take parameters. However, functions can optionally have one or more parameters.

HTTP Methods

We’ve touched on the basics of rendering templates and serving views. Now, let’s talk more about how the HTTP request methods fit into web apps. Although there are a handful of these methods, all we’ll need is the two most common:**GET** and **POST**

**The GET Method**

Any time you visit a website and a page loads in response you’re seeing the HTTP GET Method in action! The GET method is used to request (or get) information from a server. When you type in the URL of the desired website you are sending a request to that server to retrieve data which comes back as the page you see. The page is the response that has been packaged and returned to your browser in the form of HTML, CSS, and Javascript. **So far all of our routes have been for GET requsts.**

We can also use GET requests with forms when we want to pass insensitive information to a server through the URL. One common example of this is search engines, such as Google. Say we went to google, filled out the search box(which is a form), and searched for “ninjas”. The resulting URL you will see in your browser might look something like this:

<https://www.google.com/?gws_rd=ssl#q=ninjas>

Here, we sent a GET request through a search form to the Google server and sent some information with the characters following the question mark in the URL. Keeping it simple for now, that string of characters is passing information to tell the server to use a more secure version of the google website and also ninjas which is the term we searched for. Using a GET was fine for this search and is useful when we want to either bookmark or save our search in our browser history.

**POST To the Rescue**

Now to protect this information, we’ll use a POST request which passes data behind the scenes in the HTTP request message body.

These requests are never cached, not saved in your history, can’t be bookmarked and aren’t limited to how much data can be sent. The vast majority of forms you’ll be dealing with will use POST requests to send data. In general, forms should always be sent using the POST method unless you will need to give users the ability to preserve a query string, as in a Google search. Move to the next tab to see how we can apply this to Flask.

**Other HTTP Methods**

You may have noticed that we have mentioned some additional HTTP methods, PUT, PATCH and DELETE. These exist for the express purpose of designing API’s. These are not supported methods in your HTML code, and only work when being handled by JavaScript AJAX requests. This might seem strange, but we’ll learn more about why these methods exist when we talk about RESTful APIs.

**Session**

Remember that your browser makes a request to a URL and receives a response from the web server, displaying the HTML, CSS and JavaScript in your browser.

Let’s take that notion one step further. In a normal user’s interaction with a web site, that user will usually make tens, if not hundreds, of requests to a given web server.

Think about it: You’re doing your holiday shopping online. While filing your shopping cart with the gifts you’ve selected you will do the following:

* Visit the home page of your retailer of choice
* Log into your account
* Submit search queries
* Browse the items from the resulting list
* View the select items in detail
* Add certain items to your shopping cart
* Check out
* Receive confirmation of your order

Each one of those steps along the way is a full HTTP request/response cycle. It begins with your browser sending a request, and ends with the browser rendering a response from the server.

Now, here’s the thing about the HTTP request/response cycle. It is stateless. That means that each request/response cycle instance is independent and ignorant of any instances that came before or will come after it.

This matters because, along with each request, your favorite retailer always knows and remembers:

* Who you are (your account)
* What you’ve searched for
* The items in your cart
* Etc…

So how does the site know this if each HTTP request/response cycle instance is stateless?

Sites like your favorite online retailer make use of **persistent data storage!**

Persistent data storage can come in many forms, like a database, which you will learn about soon enough. It can also come in the form of writing to a file, and that’s what session does!

Now we know that a given HTTP request/response is stateless, but in the scope of a given req/res cycle, we can read certain pieces of data that we stored in previous cycles, and write certain valuable pieces of data for use in future cycles. This opens new world of user experience. With session, the user can have a conversation of sorts with a web site where a user makes decisions that can be tracked so that a server can respond appropriately to create a better user experience.

In a given process(HTTP request/response), data is created (search terms and search results for instance) that OUTLIVES the process that generated it. That data must be kept track of for use in subsequent processes. This data is called state. State allows our site to “know” a lot of useful information. Information like:

* Is there a logged in user currently?
* Who is the current logged in user?
* What links a user has viewed previously?
* Any other datapoint necessary

The last point is important. Session is a tool for you, the developer, to use to your advantage. The same way you create variables in your functions to help you solve problems, you keep state data in session to help you solve problems down the line, like in subsequent HTTP request/response cycle instances.

So persistent data storage, like session, helps us bridge the gap between a stateless protocol like HTTP with the stateful data generated through it. The combination of the two is at the heart of the modern web and heavily used by web developers around the world.

**Cookies vs Databases**

You’ve probably heard of the term cookies before. Frameworks like Flask use cookies to store data like sessions. Flask uses secure hashing of session data to send a packet of information from server to client. This packet is known as a cookie. Once your browser has received this cookie, it writes the information contained in it to a small file on your hard drive.

**Comparing/Contrasting Types Of Data Persistence**

When developers design a framework they must decide where sessions should be stored. While Flask developers decided to leverage the power of cookies for storing session data, there are certain performance and security compromises being made. Other Frameworks, like Django, choose to store session data in the database, sending only secure keys that can be used to retrieve data from a database only when required.

It is important to realize that, along with many other things in web development, we must sometimes make informed decisions about how we chose to solve a problem. When you are told something works a certain way, you should ask yourself what other solutions to that problem exist and why this solution was chosen.

Determining what data to store and where/how to store it is a subject to many considerations. System architecture, data sensitivity, data lifecycle, scope, just to name a few. Knowing more about your meta-data, that is, the data ABOUT your data as well as the differences between the common types of persistent storage is critical to architecting the optimal solution for your requirements.

|  |  |  |
| --- | --- | --- |
| **Persistent storage type** | **Lifespan** | **Common example** |
| Database | Permanent until manually deleted |  (encrypted) Passwords   (encrypted) Credit Card Info   Photos |
| Cookies | Set per browser settings |  User selected language |