



# Programming with Python

## Key Takeaways

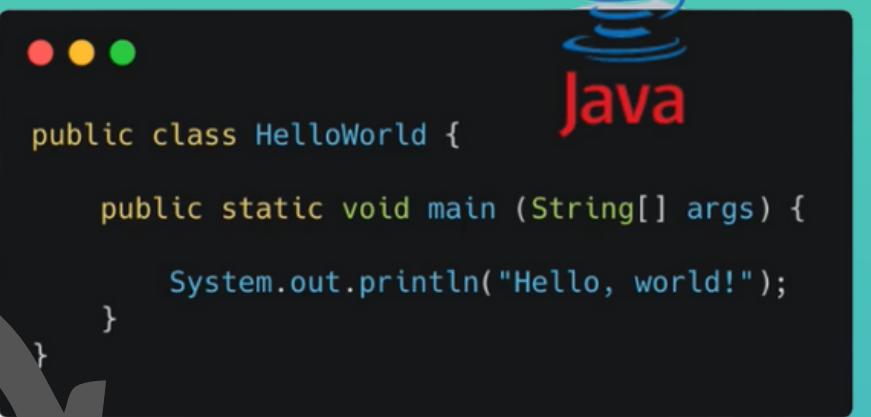
# Introduction to Python



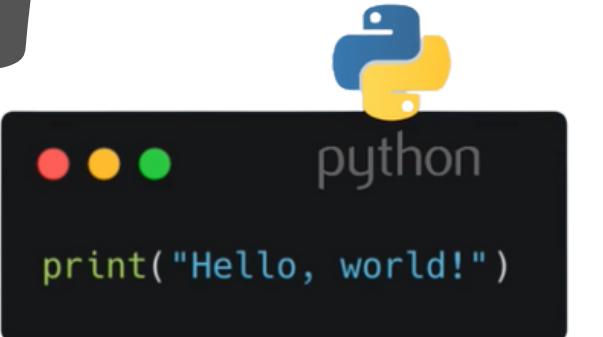
- Python is one of many **programming languages**
- The basic programming concepts are similar across the different languages

## Benefits of Python

- **Simple syntax**
- **Easy to setup**
- **Large ecosystem:** many libraries, large community
- **Flexible:** You're not limited to language specifics



```
public class HelloWorld {  
    public static void main (String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

A screenshot of a Java code editor showing a simple "Hello, World!" program. The code is written in Java syntax, using curly braces and a main method. The Java logo is visible in the top right corner of the editor window.

```
print("Hello, world!")
```

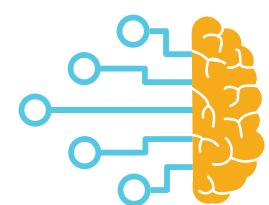
A screenshot of a Python code editor showing a simple "Hello, World!" program. The code is written in Python syntax, using a single line print statement. The Python logo is visible in the top right corner of the editor window.

- Python is one of the **most popular and widely used languages**

## What Python is used for

- Better alternatives for Mobile, Game & Desktop Development

- Web Development
- Data Science, Machine Learning, Artificial Intelligence
- Web Scraping
- **Automation:** Automate DevOps tasks and general tasks



# Why Python as a DevOps Engineer

- Knowing programming in general and Python in particular, makes you much **more valuable as a DevOps engineer**

The image shows three separate job listing snippets from a job search website, all for the position of 'DevOps Engineer'.  
1. **Hewlett Packard Enterprise** in San Jose, CA 95002: Responsibilities include managing systems, developing DevOps automation, and working with AWS. Qualifications require 4 years of experience, AWS mastery, and extensive Python experience.  
2. **Kinetix** in Irvine, CA 92618: Duties involve high technical knowledge, deep Linux expertise, and solid understanding of storage, virtualization, and CI/CD. Qualifications require 4 years of experience, AWS mastery, and extensive Python experience.  
3. **Insurify** in Sofia, NM: Duties include high technical knowledge, deep Linux expertise, and solid understanding of storage, virtualization, and CI/CD. Qualifications require 4 years of experience, AWS mastery, and extensive Python experience.

## DevOps tasks to automate with Python

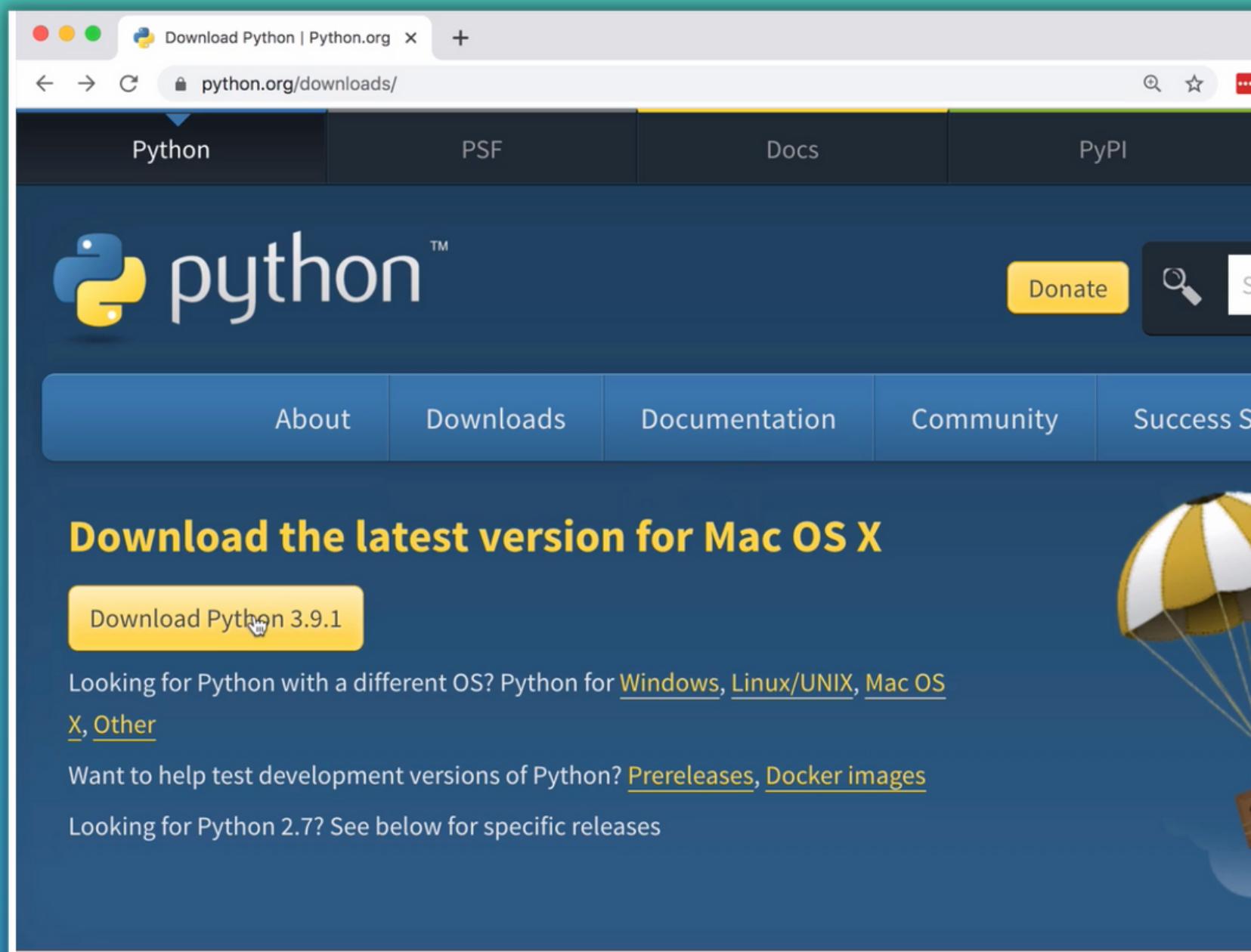
- System Health Checks
- Monitoring Tasks
- Backup Tasks
- Custom Ansible Modules
- Data Visualization
- Managing Cron
- CI/CD related Tasks (update Jira ticket after Jenkins build, trigger Jenkins job on specific events)
- Cleanup Tasks



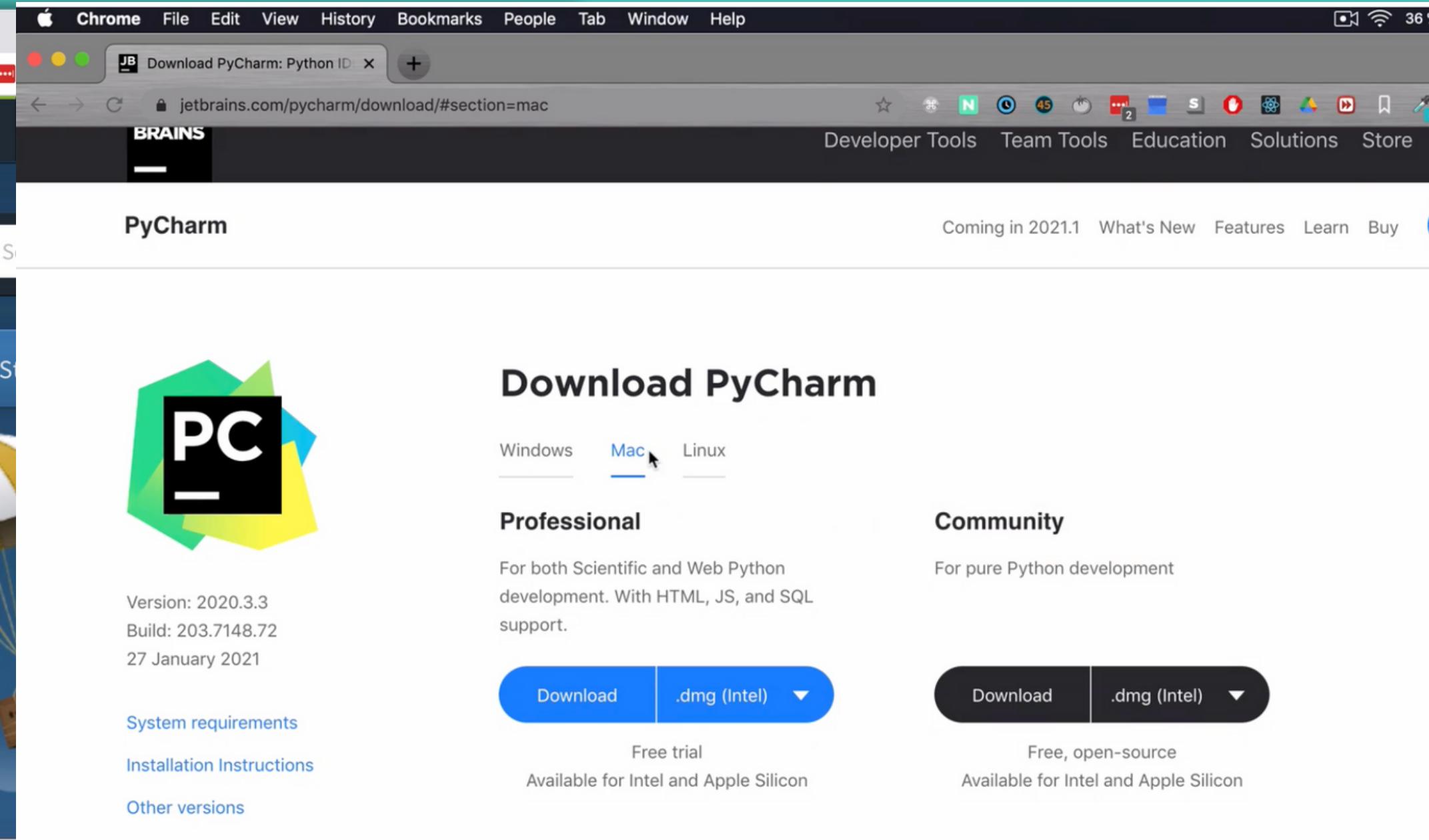
# Get Started - Local Setup

- For writing a Python program you need 2 things installed

1) Install Python version



2) Download Code Editor, like PyCharm or Visual Studio Code



# Core Programming Concepts - 1

## Variables

- Variables are **containers for storing data values**  
`score = 5`    `name = "James"`
- For repetitive values that need to be reused multiples times in different places in your code

## Comments

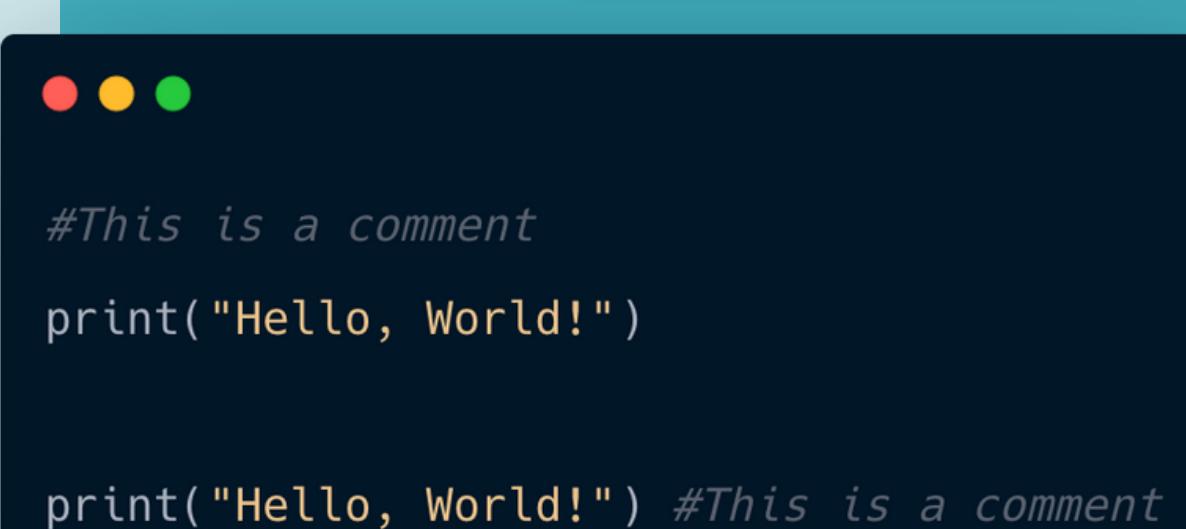
- For additional **textual description** to explain code
- Comments start with a **#**. Python will ignore it
- Usually code should be easily understandable (for example use descriptive variable names!) without comments, so don't overuse it!

## Built-in Data Types

- Variables can store data of different types
- **Different types can do different things**
- **Strings and numbers** are most basic data types

"James"      10

|                 |  |
|-----------------|--|
| Text Type:      | <code>str</code>   |
| Numeric Types:  | <code>int</code> , <code>float</code> , <code>complex</code> |
| Sequence Types: | <code>list</code> , <code>tuple</code> , <code>range</code>  |
| Mapping Type:   | <code>dict</code>  |
| Set Types:      | <code>set</code> , <code>frozenset</code>                    |
| Boolean Type:   | <code>bool</code>  |



```
#This is a comment
print("Hello, World!")

print("Hello, World!") #This is a comment
```

# Core Programming Concepts - 2

## Functions

- Function is a block of code to **perform a single, related action**
- It only **runs when it is "called"**
- For repetitive code blocks that need to be reused multiple times in different places in your code
- Functions can pass data (**parameters**) into a function
- A function can **return** data as a result of its execution

Function definition with "def" keyword

```
def sayHello():  
    print("Hello from a function")
```

sayHello()

Call the function

# Core Programming Concepts - 3

## Conditional - if/else Statements

- **Control logical flow of the program:** What to do under which condition
- **Logical conditions:**

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

## Boolean Data Type

- Boolean represent one of two values: **True** or **False**
- When you compare two values, the expression is evaluated and Python returns the boolean result



tom = 200

anna = 33

`if anna > tom:`

    print("anna has a higher score than tom")

`elif tom == anna:`     Evaluates to **false**

    print("anna and tom have an equal score")

`else:`

        Else block will be executed

    print("tom has a higher score than anna")



`print(10 > 9) => True`

`print(10 == 9) => False`

`print(10 < 9) => False`

# Core Programming Concepts - 4

## Built-in Functions

- Creators and developers of Python continuously create **functions for the most common use cases**
- So you can use these, instead of writing it yourself
- For example, accepting user input with "input"

## Accepting User Input

- In Python we can **ask the user for input**
- Here the input is saved into variable "user\_input":



```
user_input = input("Hey, enter a number of days and I will convert it to hours")
```

## Error Handling

- To **protect code** from bad user input or programming errors
- Instead of letting an error crash the program, **write code to handle the error** when it happens
- In below example, if user doesn't enter a number, it will crash when we do the calculation. So we would need to handle this case, for example with if/else statement

# Core Programming Concepts - 5

## Lists

- Are used to **store multiple items** in a single variable
- Can contain different data types
- Items are **indexed**, the first item has index [0]
- List items are ordered and **allow duplicate values**

```
friends = ["tim", "sara", "john"]
```

- You can loop through the list items by using a "for loop"

```
● ● ●  
friends = ["tim", "sara", "john"]  
for friend in friends:  
    print(friend)
```

## Loops

- Allows us to **execute a statement multiple times**
- Different types of Loops

### For Loop:

- Logic that should repeat FOR a specific number of times or **FOR a set of values**

### While Loop:

- For logic that should **repeat as long as a certain condition is true**, like WHILE a user is shopping, we will keep the shopping cart active

```
● ● ●  
count = 0  
while (count < 9):  
    print 'The count is:', count  
    count = count + 1  
  
print "Good bye!"
```

# Core Programming Concepts - 6

## Sets

- Similar to list, as you can store multiple items in a set
- Difference: You **can NOT have duplicate values** and it's **unindexed**
- Good to keep track of lists that must have unique values, like student ids in a university database

```
studentIds = [3441, 1661, 7845]
```

- You can also loop through a set

## Dictionaries

- More complex data structure that groups multiple simple data types together
- Stores data values in **key:value pairs**
- Duplicates are not allowed

```
● ● ●  
aPerson = {  
    "name": "Sara",  
    "languages": ["python", "javascript"],  
    "age": 29  
}  
  
print(aPerson)
```

# Modules - 1

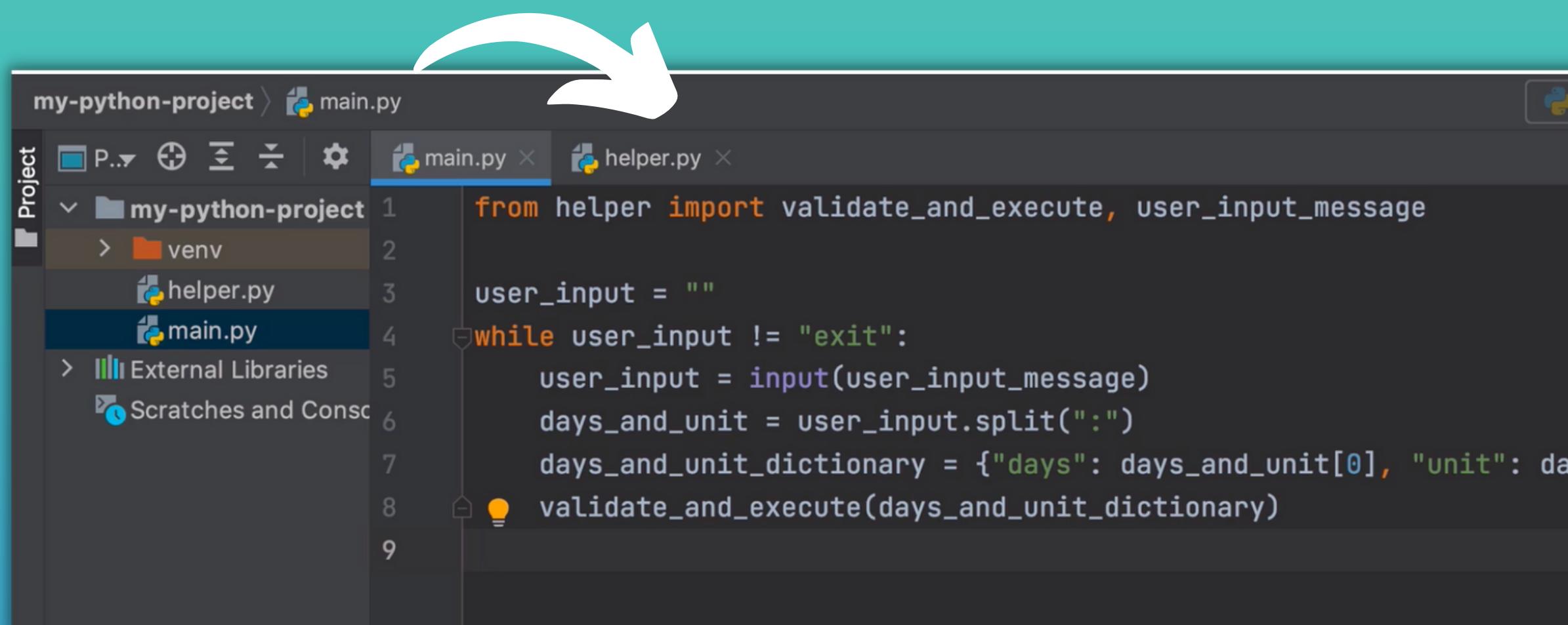
- A module allows you to **logically organize your Python code**, so it should contain related code
- Grouping related code into a module **makes the code easier to understand and use**

## Create a Module

- It's just a .py file consisting of Python code. For example *helper.py*

## Use a Module

- To use the module, we need to **import** it



```
my-python-project > main.py
Project  P... + - ⚙️ ⚙️ main.py × helper.py ×
my-python-project
  v my-python-project
    > venv
    helper.py
    main.py
  > External Libraries
  Scratches and Consoles

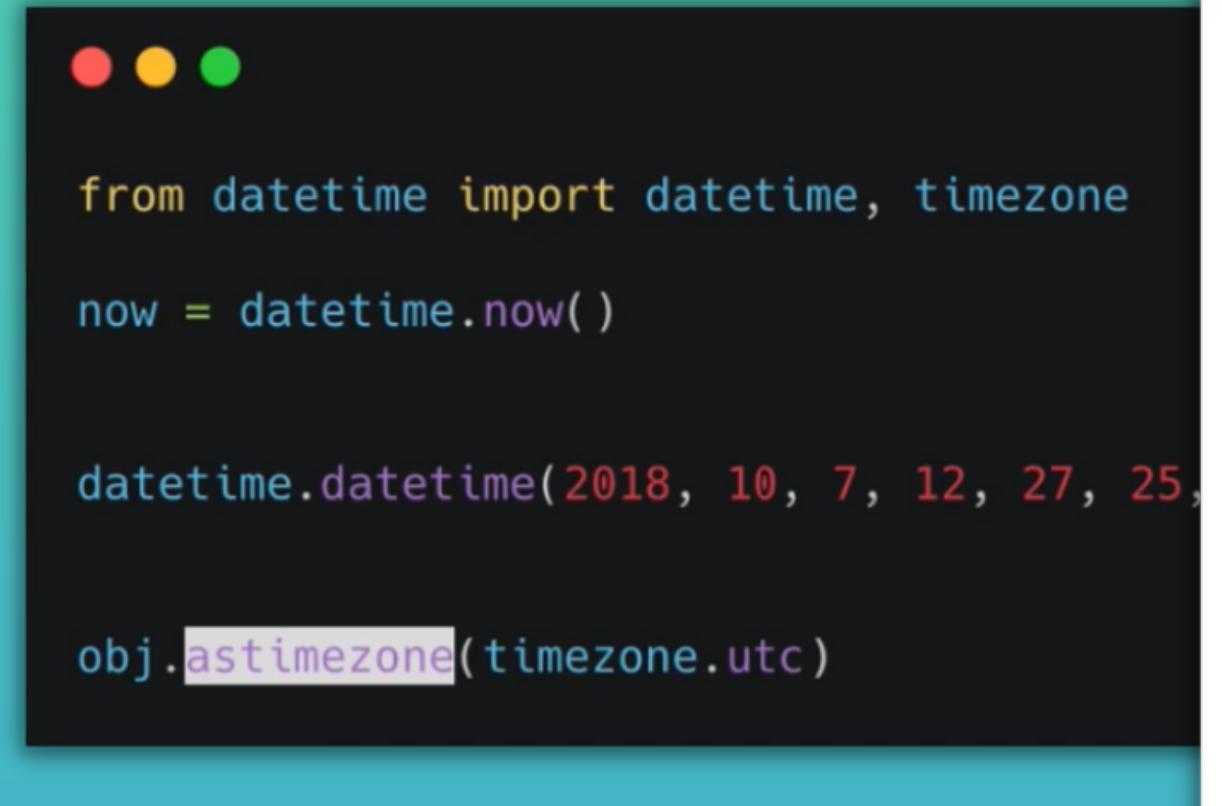
1   from helper import validate_and_execute, user_input_message
2
3   user_input = ""
4   while user_input != "exit":
5       user_input = input(user_input_message)
6       days_and_unit = user_input.split(":")
7       days_and_unit_dictionary = {"days": days_and_unit[0], "unit": da
8       validate_and_execute(days_and_unit_dictionary)
9
```

# Modules - 2

- You can also use **existing modules**
- Many useful ones, for example  
datetime module which have many  
different functions that make it  
**easier for you to work with dates  
and times!**

## Built-In vs Third-Party

- Python comes only with a set of built-in modules
- Many more available, which are NOT part of the Python installation
- You need to install these **third-party packages**
- For example you may need specific ones for writing a web application or a machine learning app



```
from datetime import datetime, timezone
now = datetime.now()
datetime.datetime(2018, 10, 7, 12, 27, 25,
obj.astimezone(timezone.utc)
```

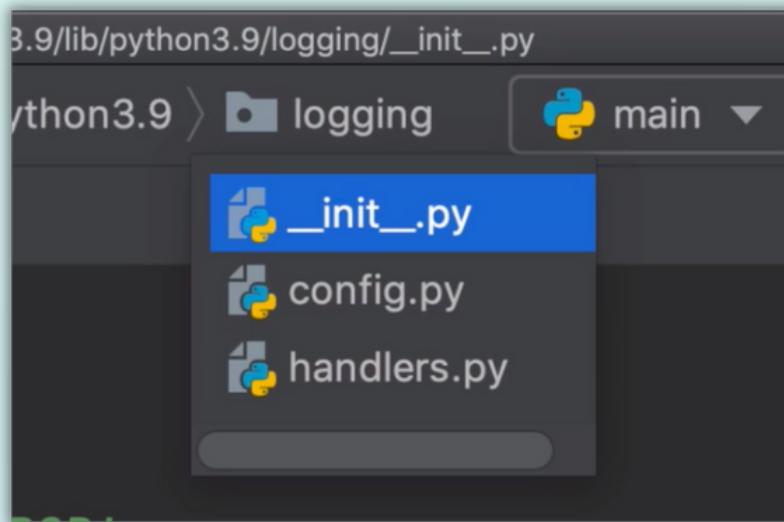
## Python Module Index

| <a href="#">_</a> |  |
|-------------------|--|
| <a href="#">a</a> | <a href="#">future</a>   |
| <a href="#">b</a> | <a href="#">main</a>   |
| <a href="#">c</a> | <a href="#">thread</a>   |
| <a href="#">d</a> |  |
| <a href="#">e</a> | <a href="#">Future statement definitions</a>   |
| <a href="#">f</a> | <a href="#">The environment where top-level code is run. Covers command-line interfaces, import-time behavior, and ``__name__ == '__main__'``.</a> |
| <a href="#">g</a> | <a href="#">Low-level threading API.</a>   |
| <a href="#">h</a> |  |
| <a href="#">i</a> | <a href="#">Abstract base classes according to :pep:`3119`.</a>  |
| <a href="#">j</a> | <a href="#">Read and write audio files in AIFF or AIFC format.</a>   |
| <a href="#">k</a> | <a href="#">Command-line option and argument parsing library.</a>  |
| <a href="#">l</a> | <a href="#">Space efficient arrays of uniformly typed numeric values.</a>  |
| <a href="#">m</a> | <a href="#">Abstract Syntax Tree classes and manipulation.</a>   |
| <a href="#">n</a> | <a href="#">Support for asynchronous command/response protocols.</a>   |
| <a href="#">o</a> | <a href="#">Asynchronous I/O.</a>  |
| <a href="#">p</a> | <a href="#">A base class for developing asynchronous socket handling systems.</a>  |
| <a href="#">q</a> | <a href="#">Register and execute cleanup functions.</a>  |
| <a href="#">r</a> | <a href="#">Manipulate raw audio data.</a>   |
| <a href="#">s</a> |  |
| <a href="#">t</a> | <a href="#">RFC 4648: Base16, Base32, Base64 Data Encodings; Base85.</a>   |
| <a href="#">u</a> | <a href="#">Debugger framework.</a>  |
| <a href="#">v</a> | <a href="#">Tools for converting between binary and various ASCII-encoded representations.</a>   |
| <a href="#">w</a> | <a href="#">Encode and decode files in binhex4 format.</a>   |
| <a href="#">x</a> | <a href="#">Array bisection algorithms for binary searching.</a>   |
| <a href="#">y</a> | <a href="#">The module that provides the built-in namespace.</a>   |
| <a href="#">z</a> | <a href="#">Interfaces for bz2 compression and decompression.</a>  |

# Packages, PyPI and pip

## Package

- **Module** = Any Python file is a module
- **Package** = A collection of Python modules
- Package must include an `__init__.py` file
- This file distinguishes a package from a directory

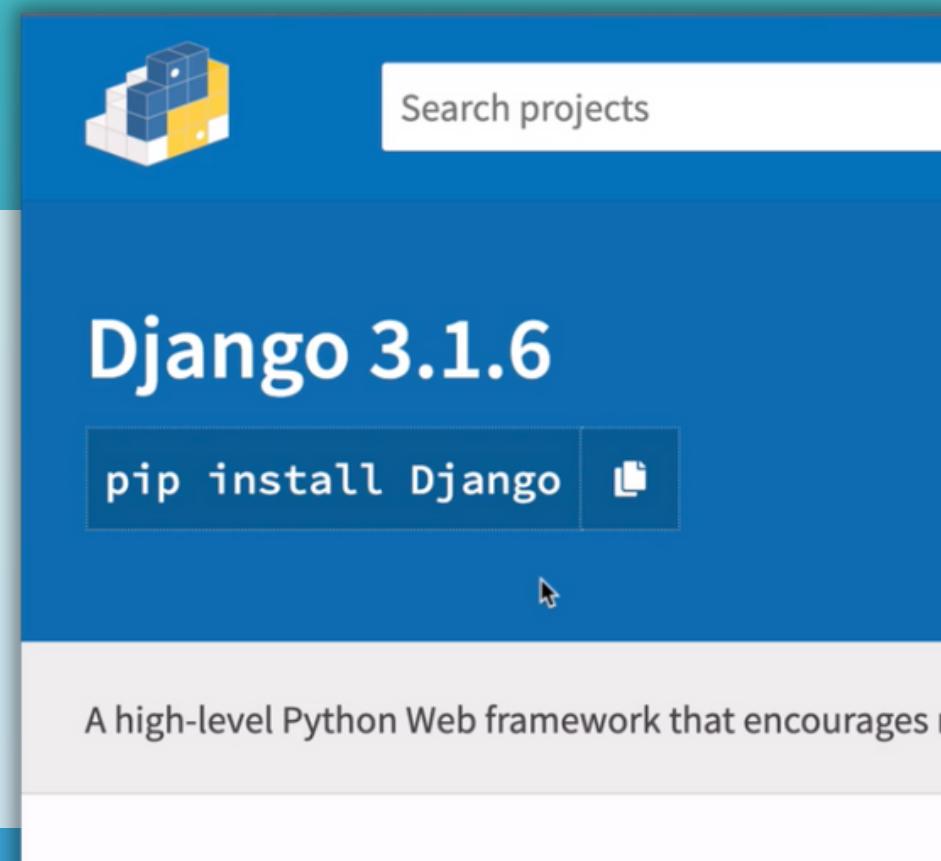


## pip

- **Package installer** for Python
- Used to **install packages** from the Python Package Index , but also other indexes

## PyPI - Find Packages

- **Repository for third-party Python packages**
- People can publish their packages to this repository
- So it becomes **available for everyone to use**
- Large Community means, many people are creating useful modules and make them available for others



# Object-oriented Programming - 1

- Python is a object oriented programming language
- Almost everything in Python is an object, with its properties (or also called attributes) and methods (functions that belong to the object)

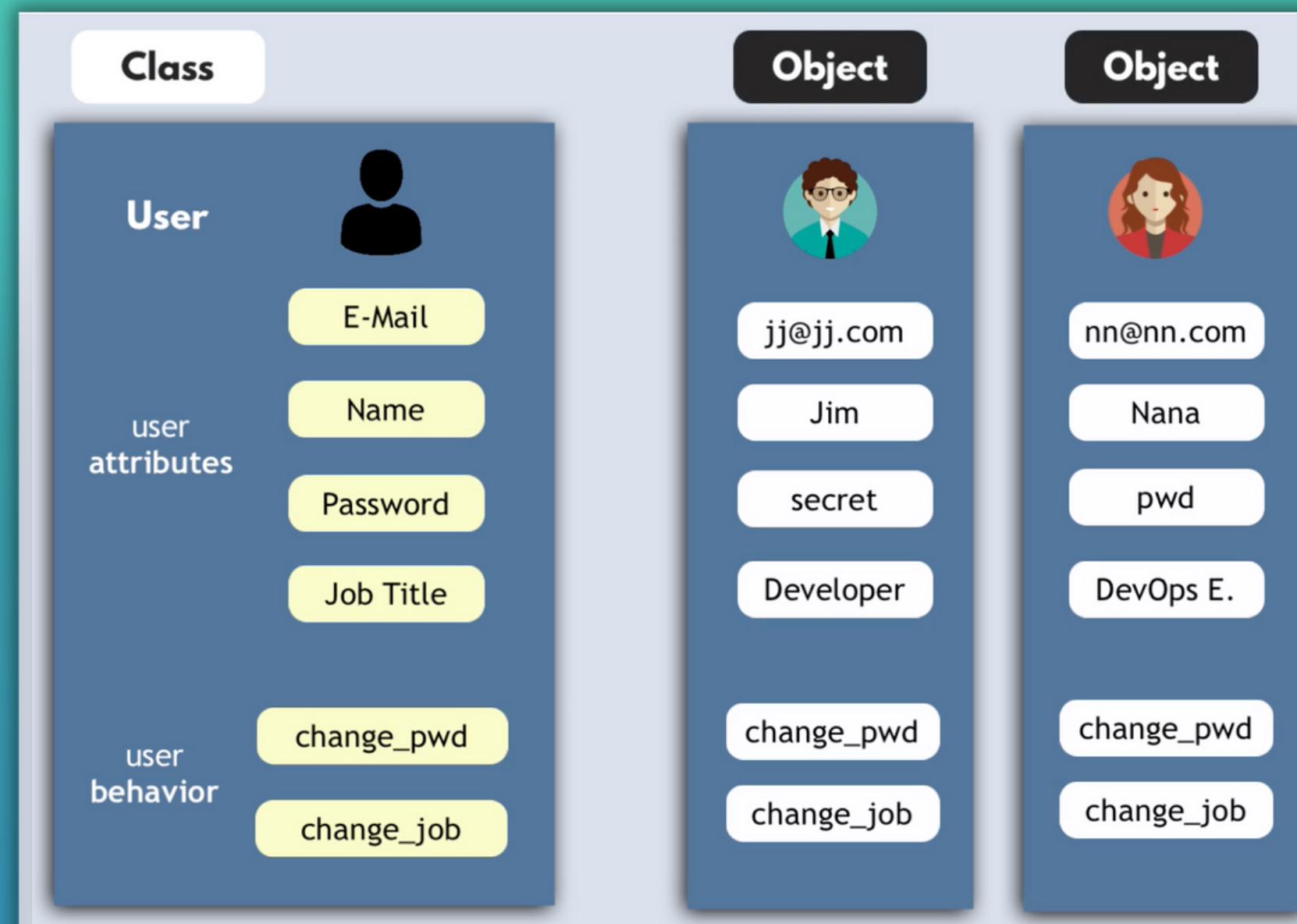
**Class**

• Like a object constructor or a **blueprint for creating objects**

**Object**

• Object is a unique **instance** of that class

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
p1 = Person("Jim", 32)  
  
print(p1.name)  
print(p1.age)
```



# Object-oriented Programming - 2

- All classes have a function called `__init__()`, which is always **executed when the class is being initiated.**
- Use the `__init__()` function to **assign values** to object properties, or other operations that are necessary to do when the object is being created



```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    p1 = Person("Jim", 32)  
  
    print(p1.name)  
    print(p1.age)
```

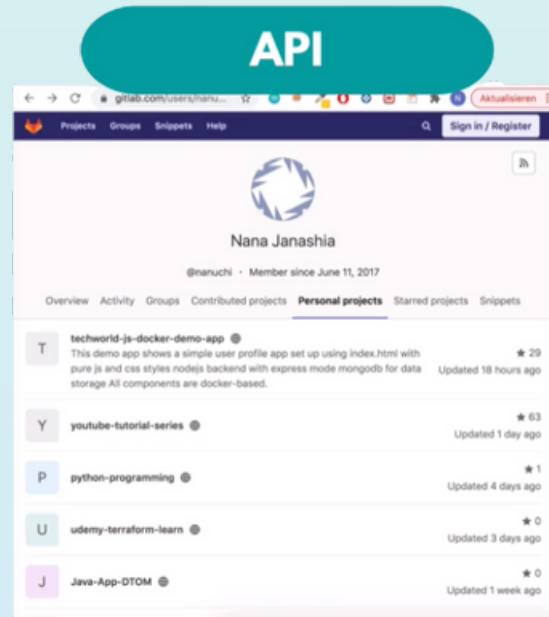
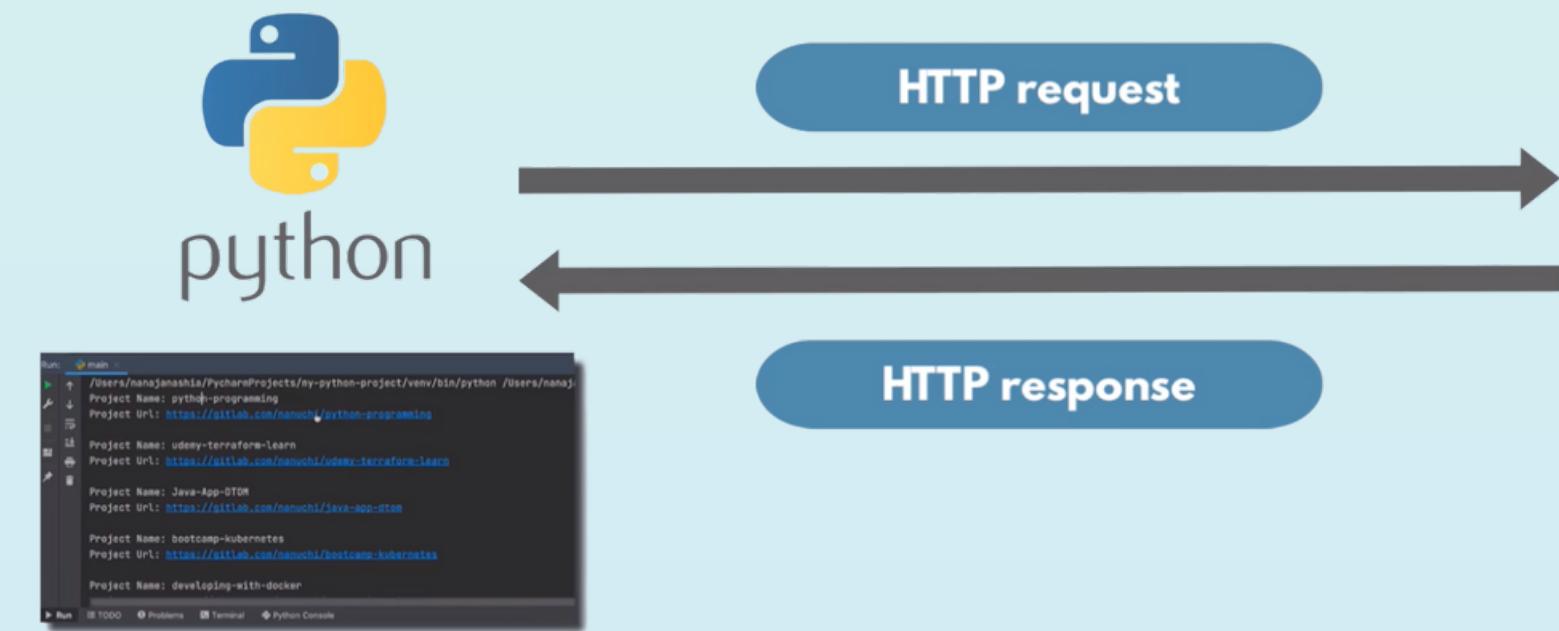
# API Requests - 1

## Application Programming Interface (API)

- API is code layer of an application that **lets other applications talk to it**.
- So it's used as a **communication interface** between that app (e.g. Gitlab) and any other apps that want to talk to it.

## HTTP Request and Response

- To communicate, the app (python app) must send an "**API request**" to GitLab, like "Hey GitLab, give me a list of repositories for user xx" and GitLab will respond with the reply ("**API response**").



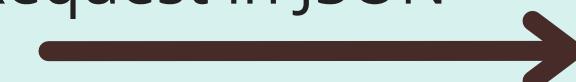
# API Requests - 2

JSON

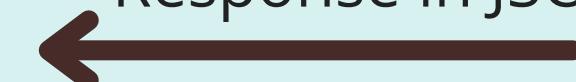
- Lightweight **format for transporting data**
- Often used to send data over the web



Request in JSON



Response in JSON



A screenshot of a GitHub profile page for 'Nana Janashia'. The page shows a profile picture, the user's name, and a list of personal projects. The projects listed are: 'techworld-js-docker-demo-app' (updated 18 hours ago), 'youtube-tutorial-series' (updated 1 day ago), 'python-programming' (updated 4 days ago), 'udemy-terraform-learn' (updated 3 days ago), and 'Java-App-DTOM' (updated 1 week ago). The 'Personal projects' tab is currently selected.

How to do it in Python code?

A screenshot of a Python code editor showing a file named 'main.py'. The code imports the 'requests' library and makes a GET request to 'https://gitlab.com/api/v4/'.

```
1 import requests
2
3 requests.get("https://gitlab.com/api/v4/")
```

"**requests**" = simple HTTP  
library for *sending requests*

A screenshot of a Python code editor showing a file named 'main.py'. The code imports 'requests' and uses it to get data from 'https://gitlab.com/api/v4/'. It then prints the JSON response and the type of the response text.

```
1 import requests
2
3 response = requests.get("https://gitlab.com/api/v4")
4 print(response.json())
5 print(type(response.text))
```

"**json()**" = function converts the data  
from json into Python data type

# API Requests - 3

## Authenticate

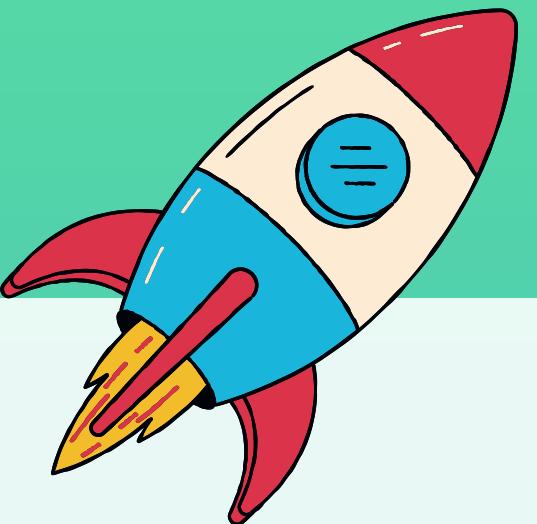
- Many web services require authentication to access for example personal data, like private repositories of users etc.
- For that the service (GitLab for example) must request validation that it's the user herself requesting the private information, and not some other user, who is not authorized to see that information.
- So in the request, application (python program) must send credentials too, which API will validate.
- There are different authentication types



Basic Authentication

```
requests.get('https://gitlab.com/api/v4/jim/repositories', auth=('user', 'pass'))
```

# Best Practices - 1



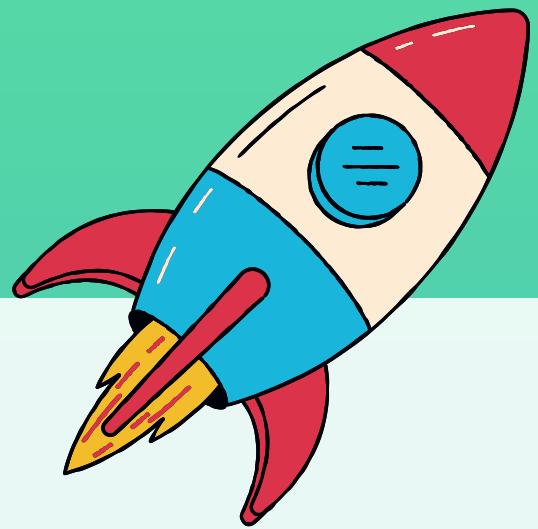
## Naming Conventions for Variables and Function Names:

- Use **descriptive variable names!**
- Use lowercase and underscores to separate words
- Class file should be lowercase, whereas the class name itself should start with an uppercase.  
Examples: user.py file, but class User
- Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples: MAX\_OVERFLOW and TOTAL

## Comments:

- **Don't overuse comments!** The program itself with descriptive variable and function names should already be very descriptive.
- Only use comments, when it adds more information. Example: `a + b`, bad would be to write a comment here: `# here I add a plus b`

# Best Practices - 2



- Use Spaces for Indentation

## Functions:

- **DRY - Don't Repeat Yourself** - a general principle of software development intended for reducing repetition. Whenever you code something more than once you can actually create a function for the logic or a variable for the value.
- Functions should be small and encapsulate one single functionality. If your function needs many parameters, you probably include too much logic and the function should rather be splitted. With this approach you will also have no problem in naming the function and it's easier to reason about the program.

Official Style Guides for Python: <https://www.python.org/dev/peps/pep-0008/>