esade

Advanced Python

Class 2

Pepe Bonet Giner

10th Jan 2024

Index Class 2

Topic 1: Recap Class 1

Topic 2: Python CLI Parsers. Click

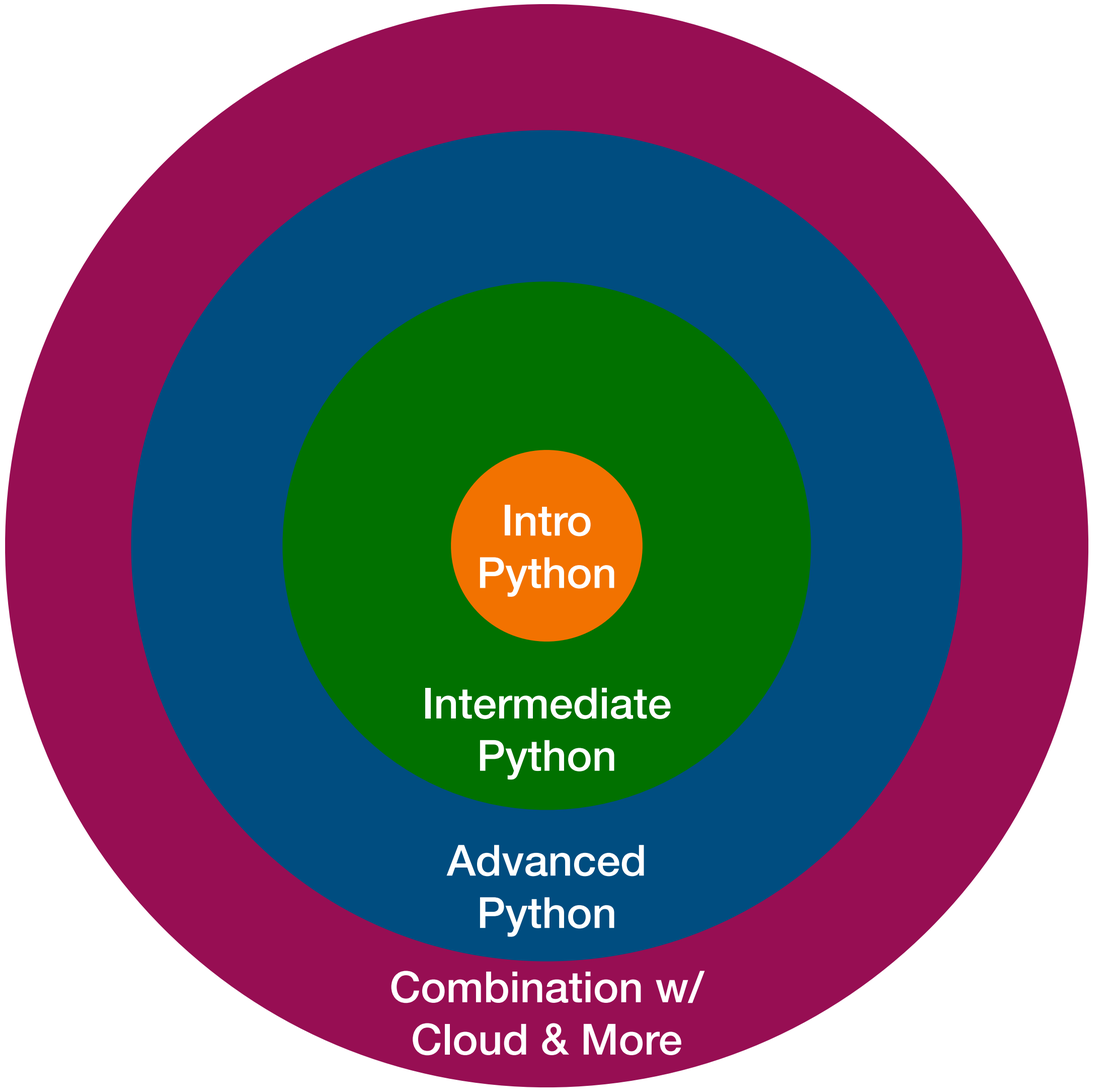
Topic 3: Debugging. Try & Except

Topic 4: Python Classes

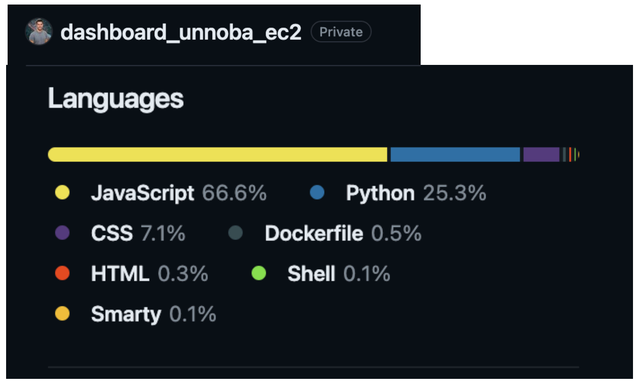
esade

Topic 1: Recap Class 1

Advanced Programming/Python



- 1 Person
- 2-10 People
- 100s People
- 1000s or M People



The Development Environment

"If you have time to set up only one piece of infrastructure well, make it the development environment"

Ville Tuulos

IDE

- Editor where you write the code

Code Versioning

- Store & track different versions and stages of your code

CI/CD

- Continuous Improvement & Continuous Development

VSCode + Environment + Command Line = Good Start

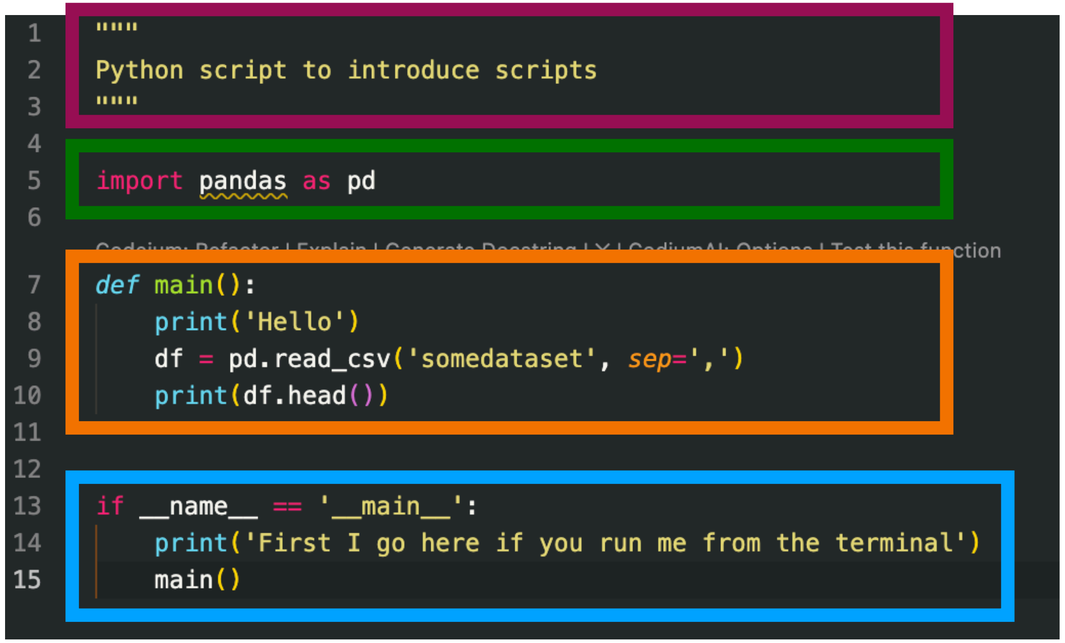

```
Django>=4.0.1,<4.1
djangorestframework>=3.13.1,<3.14
psycopg2>=2.9.3,<2.10
drf-spectacular>=0.22.1,<0.23
Pillow>=9.1.0,<9.2
uwsgi>=2.0.20,<2.1
```

```
echo "Step 2: Downloading the backup..."
scp root@guarani-cisa.unnoba.edu.ar:~/base.backup .

#Restore backups
echo "Step 3: updating the .pgpass file"
python write_pgpass.py
chmod 600 ~/.pgpass

echo "Step 4: clear database for restore..."
#Run script to update the files
python3 clear_database.py
```

Python Scripts

Scripts are small programs in Python that allow us to automate tasks

- **Docstring:** Explains what we do in the script
- **Packages (Libraries)**: That we will use in the code
- **Main Function:** Where the code will run.
- **Python Idiom:** To check whether the script is run from the terminal



esade

Topic 2: Python Parsers. Click

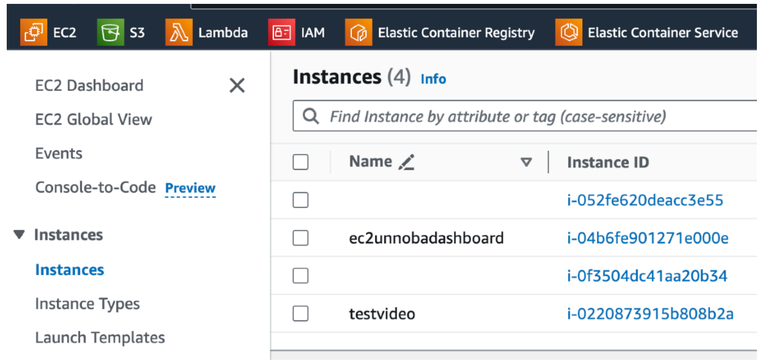
What is a CLI?

A command-line interface (CLI) is a text-based user interface used to interact with software and operating systems by typing commands into a console or terminal.

CLI

```
C:\>aws ec2 terminate-instances --instance-ids i-04608b17e51a1400c
{
    "TerminatingInstances": [
        {
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "InstanceId": "i-04608b17e51a1400c",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

GUI



Why are they useful?

- **User-Friendly Interfaces**: Creation of CLIs for Python applications.
- **Argument Handling**: They manage and interpret command-line arguments and options, making it easier to control program behavior from the command line.
- **Automation Support:** Facilitates the automation of tasks via scripts and command-line tools.
- **Error Handling & Help**: Offers built-in error handling for invalid arguments and ca automatically generate help and usage messages.

Python Options

Usage Example

Some Ideas

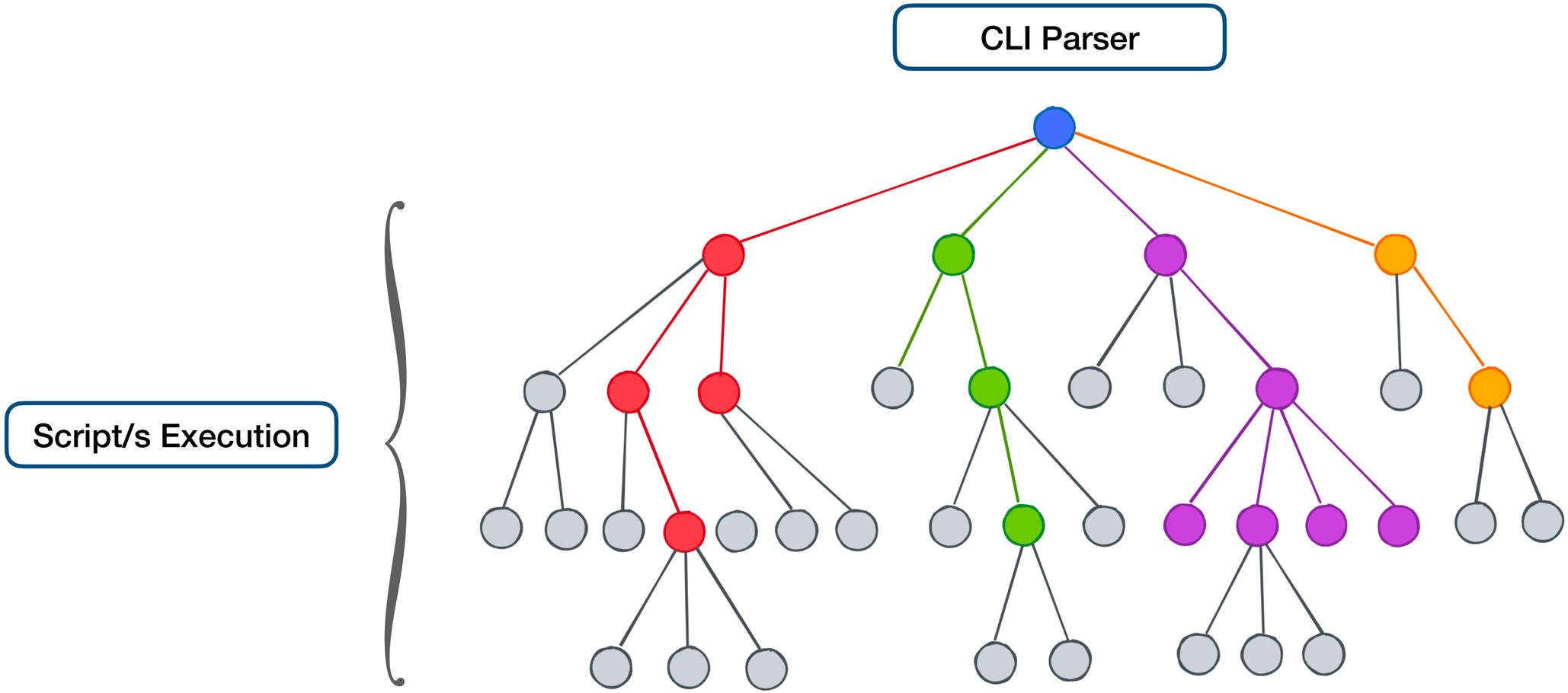
- Apply filters from the command line with a flag
- Different splits for train-test in an ML model
- Use different Models
- Apply different ways of cleaning the data

(All of this from the command line)

DeepMP

- https://github.com/pepebonet/DeepMP/blob/release/deepmp/DeepMP.py

This came to my mind while preparing it



Building blocks when using Click (our CLI parser choice)

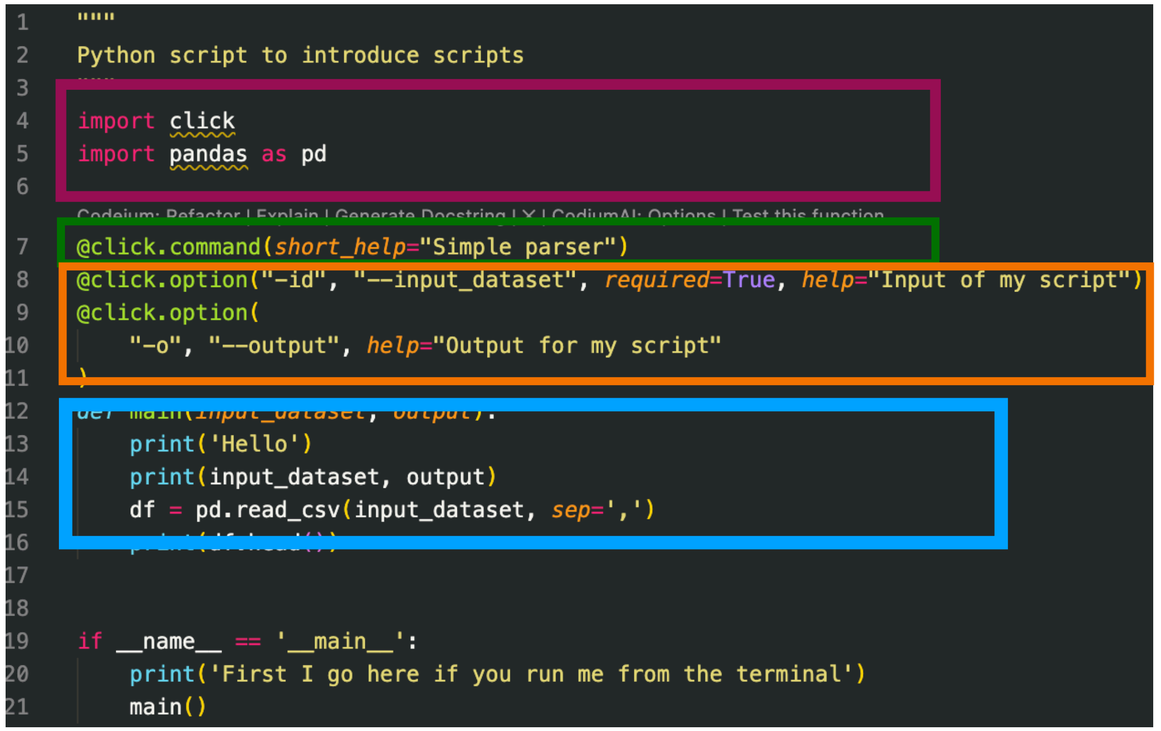
- **Library:** Explains what we do in the script
- **click.command**: That we will use in the code
- **click.options:** Where the code will run.
- **Pass it to function:** Pass to main and use them in the function

```python
"""
Python script to introduce scripts
"""

import click
import pandas as pd


@click.command(short_help="Simple parser")
@click.option("-id", "--input_dataset", required=True, help="Input of my script")
@click.option(
    "-o", "--output", help="Output for my script"
)
def main(input_dataset, output):
    print('Hello')
    print(input_dataset, output)
    df = pd.read_csv(input_dataset, sep=',')
    print(df.head())


if __name__ == '__main__':
    print('First I go here if you run me from the terminal')
    main()
```

- **Let's build a Python CLI Parser**

Exercise

- Build a simple Python CLI Parser in our Python script using click
- We need to install the package, import it, and use it properly.



```python
"""
Python script to introduce scripts
"""

import click
import pandas as pd


Codeium: Refactor | Explain | Generate Docstring | X | CodiumAI: Options | Test this function
@click.command(short_help="Simple parser")
@click.option("-id", "--input_dataset", required=True, help="Input of my script")
@click.option(
    "-o", "--output", help="Output for my script"
)
def main(input_dataset, output):
    print('Hello')
    print(input_dataset, output)
    df = pd.read_csv(input_dataset, sep=',')
    print(df.head())


if __name__ == '__main__':
    print('First I go here if you run me from the terminal')
    main()
```

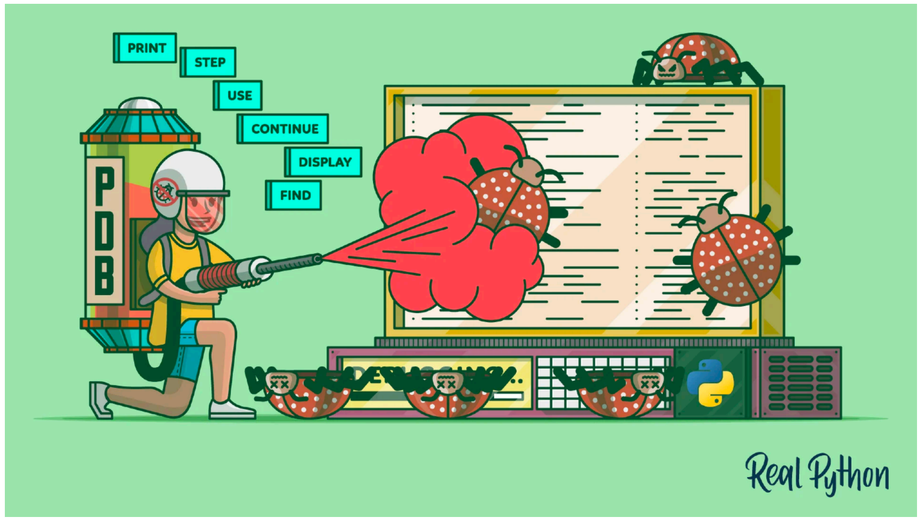
esade

Topic 3: Debugging.
Try & Except

Python Debuggers

Debuggers are tools or utilities that help programmers identify and resolve bugs (errors) in their code

- **Error Identification:** Pinpointing the exact location of errors in the code.
- **Code Understanding:** Understand how a program operates
- **Efficiency:** Speeds up the process of fixing bugs compared to manual debugging methods (like print statements).
- **Quality Assurance:** Improve code quality by facilitating inspection and testing.
- **Learning Tool**



Debugger Intuition

No Debugger

Code Working

Code Working

Code Working

Code Working

Code Not Working.

More Code

More code

Debugger

Code Working

Code Working

Code Working

Code Working

Add debugger (stop the code)

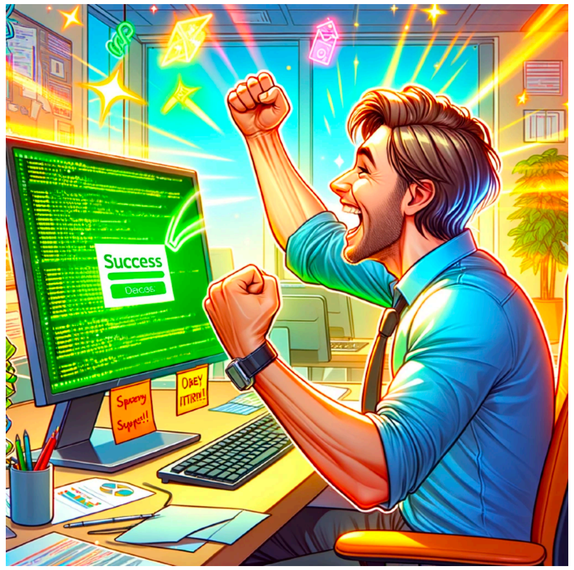
Inspect the problem

Solve it

Code Not Working —> Code Working

More Code

More code



Python Debugger at work

- **Debugger:** Stop the code and maybe inspect why the code below is giving problems
- Stops the code at that line and allows you to access all variables and the state of the program at that point
- You control it from the command line

```
def main(input_dataset, output, operation, filters):
    """
    Main function to start inspecting the table
    """
    df = pd.read_csv(input_dataset, sep=',')
    import pdb;pdb.set_trace()

    if filters:
        df = df[df['Publish Date (Year)'] > 2015]

    if operation == 'mean':
        print('I am in the mean')
        print(df['Price Starting With ($)'].mean())
    else:
        print('I am in the median')
        print(df['Price Starting With ($)'].median())
```

There are other options (IDE-specific debuggers) but **we will use pdb**

Exercise

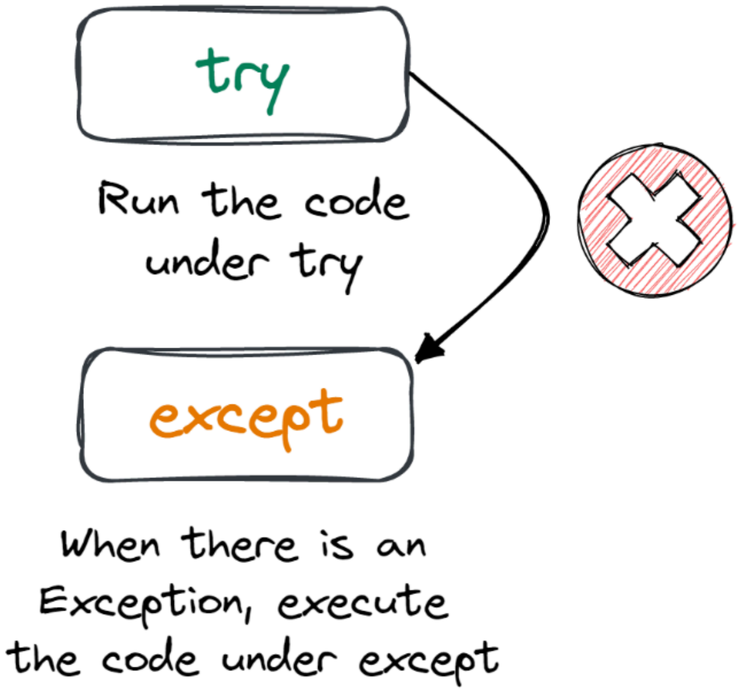
Let’s see it in action!

Use pdb as our python debugger and run commands from the command line

We just need to add: import pdb;pdb.set_trace()

Python *try* and *except*

try and *except* are keywords used for exception handling. This mechanism allows a program to continue execution even if an error occurs in a specific block of code.



Python *try* and *except* - *Usefulness*

- **Error Handling:** Allows a program to handle errors gracefully without crashing.
- **User Experience:** Improves the user experience by providing meaningful error messages instead of cryptic stack traces.
- **Robustness:** Increases the robustness of the application by enabling it to cope with unexpected situations.
- **Debugging Aid:** Assists in debugging by catching and identifying exceptions.

try

except

```python
def main(input_dataset, output, operation, filters):
    """
    Main function to start inspecting the table
    """
    try:
        df = pd.read_csv(input_dataset, sep=',')
    except:
        raise Exception("An error occurred while reading the file. Please try again")
```

There are a variety of exceptions

Simple Exception

```
try:
    something
except Exception as e:
    raise Exception(f"There was some problem: {e}")
```

ValueError

```
try:
    int('abc')
except ValueError as e:
    raise ValueError(f"Caught a ValueError: {e}")
```

FileNotFoundError

```
try:
    df = pd.read_csv(input_dataset, sep=',')
except FileNotFoundError as e:
    raise FileNotFoundError(f"Error Reading the file: {e}")
```

TypeError

```
try:
    'a' + 5
except TypeError as e:
    raise TypeError(f"Caught a TypeError: {e}")
```

Exercise

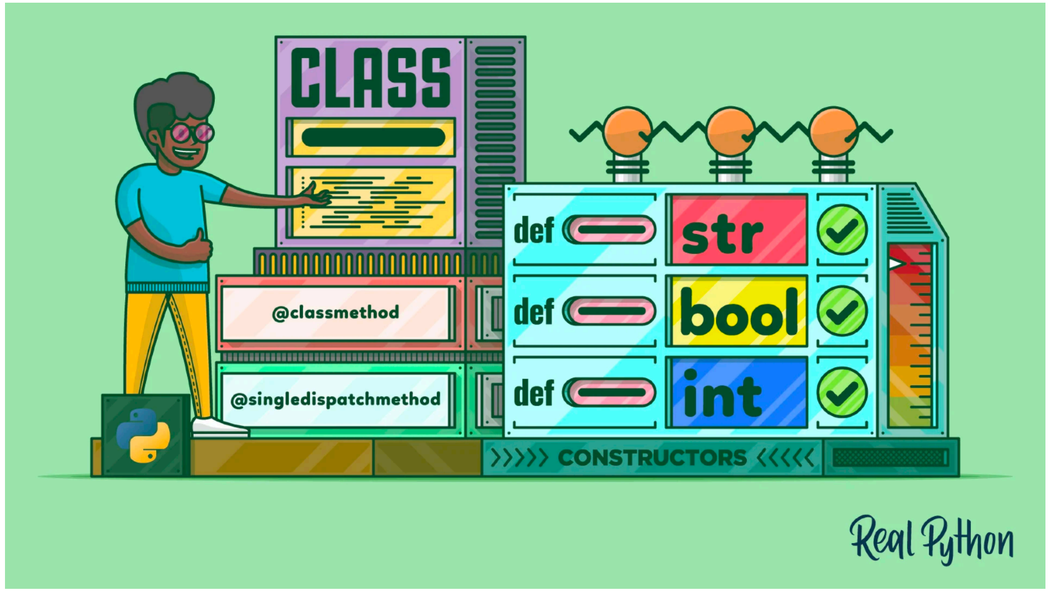
Let’s build some try and except statements!

esade

Topic 4: Python Classes

Python Classes

Classes are a code template for **creating objects** (a particular data structure), **providing initial values** for state (member variables or attributes), and i**mplementations of behavior** (member functions or methods). Classes encapsulate data for the object and the methods to interact with that data.



Why Classes?

- **Encapsulation:** Classes encapsulate data and functions, making code more modular and organized.
- **Code Reusability:** Reducing redundancy and making maintenance easier.
- **Inheritance and Polymorphism:** Classes allow the use of inheritance and polymorphism (same name different executions), key concepts in object-oriented programming, enabling flexibility and code reuse (Not always easy to grasp).

Data Modelling

GUI & Web Applications

Game Development

Main Building Blocks of Classes

- **Constructor:** Method to start the class and create objects
- **Attributes**: Variables that hold the specific objects
- **Methods:** Functions that define behaviors and can manipulate attributes.
- **Class Calling:** Wat to call classes and create objects

```python
class Dog:
    # Constructor
    def __init__(self, name, age):
        self.name = name  # Attribute
        self.age = age    # Attribute

    # Method
    def bark(self):
        return "Woof!"

# Creating an object of the Dog class
my_dog = Dog("Buddy", 5)
print(my_dog.bark())  # Output: Woof!
```

Exercise

- Let’s build our first class and bring things done today together

```
Codeium: Explain | CodiumAI: Options | Test this class
class filters_dataset:

    Codeium: Refactor | Explain | Generate Docstring | ×
    def __init__(self, df):
        self.df = df

    Codeium: Refactor | Explain | × | CodiumAI: Options | Test this method
    def filter_publication_year(self, year):
        """
        Filter books by year
        """
        return self.df[self.df['Publish Date (Year)'] == year]
```

Exercise

- Do all we have done with your own code, a bit different, and with some other examples
- Can you do the following?
  - Create a command for the CLI that takes a price value and if the command “filters” is true and we have a price value, it takes the dataset and filters books that are more expensive than the price, but the filter is in a class with other filters.
  - Put the year and month as options for filtering