GU ROCKETRY PRESENTS

CODING WITH ROCKETS

PART 1

INTRODUCTION AND VARIABLES

To introduce you to coding, we're going to use the Python programming language!

Link to Repl:

https://replit.com/@gurocketry/Coding-with-Rockets

When you see a line of code beginning with #, this is a comment.

- The computer ignores comments because they are only for us to see
- We can use comments to describe our code. Here, we will use them to help you understand the programs.

When we code, we use "Variables" to store and manipulate data. They can hold values, like words and numbers.

In Python, we can declare variables like this:

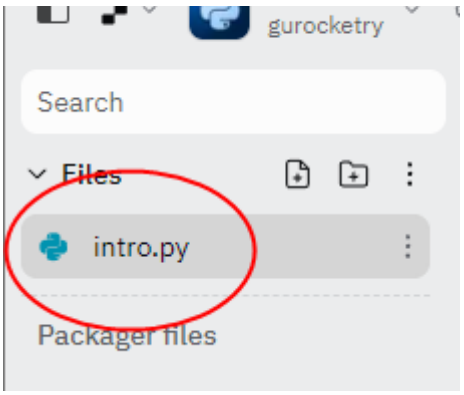
```
variable_name = value
```

Here, `variable_name` is the name of the variable, and `value` is the information we are storing inside of it.

We can picture the variable as a box, which contains data. We can change what the box holds (or the "*value*") inside by using the equals symbol (=).

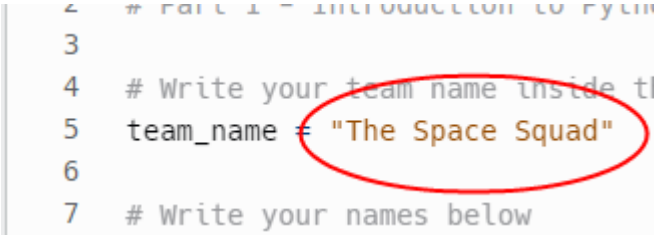
TRY IT OUT

First, click on "intro.py".



Then, modify the value of the `team_name` variable so it holds the name of your team.

For example, if your team is called The Space Squad, your code should look like this:



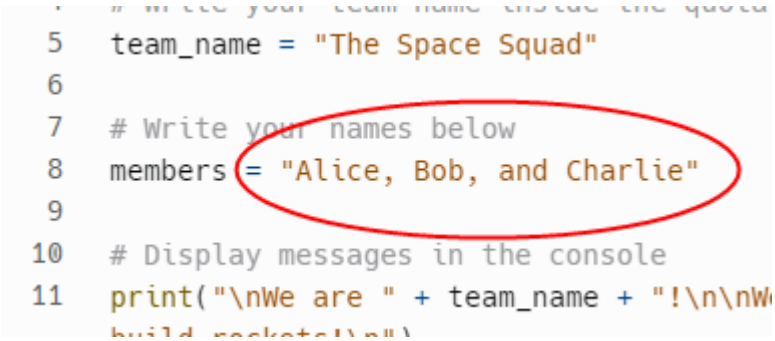
Now, click the ▶ **Run** button at the top of the screen to run your code.

In the *Console* window, you should see a message displaying your team name!

NEXT TASK

Now, let's change the members variable to contain your names.

For example, if your team consists of Alice, Bob, and Charlie, your code should look like this:



```
5  team_name = "The Space Squad"
6
7  # Write your names below
8  members = "Alice, Bob, and Charlie"
9
10 # Display messages in the console
11 print("\nWe are " + team_name + "!\n\nW
```

OPTIONAL TASKS

- Try printing out more messages to the console! At the bottom of the code, use `print()` with a message in quotations (“”)
- Create a new variable using the `variable_name = value` format
- Display your new variable using `print()`!

TIPS

- Make sure you include your team name in quotations, otherwise, the code won’t work!

Congratulations! You can now use *variables*!

PART 2

USING FUNCTIONS

One of the most important ideas in programming is “functions”.

- Functions are *blocks of reusable code*, which can be *called* at any time and as many times as needed.

Below is an example of a function:

```
def greeting():
    print(“Hello world!”)
```

We **define** a function called “greeting” with the `def` keyword, followed by brackets **()** and a colon **:**

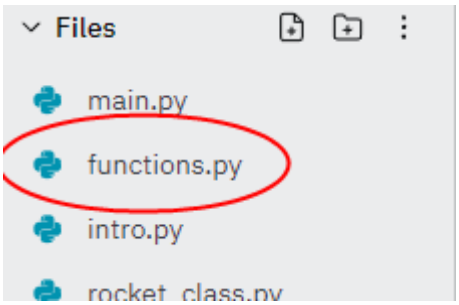
- *Inside* the function, we write the code we want to run when we *call* the function (e.g. display “Hello world!” to the console)

When we want to run the function, we “call” it with `greeting()`.

- If we don’t call a function, then the code inside it will not run
- We can also call functions multiple times

LET’S SEE THEM IN ACTION!

Switch to functions.py.



Click the ▸ **Run** button and look at the console.

Once the program has finished, have a look at the code. You should be able to see four functions:

```
- prepare()
- launch()
- navigate_to()
- land_on()
```

At the bottom of the code, you should see that we call each of these functions one after the other.

NOW IT'S YOUR GO!

Near the bottom of the code, find the `planet` variable:

```
- planet = "Mars"
```

Change this to a different planet:

- Do you remember the planets in our solar system? Can you name them all?
- ▸ **Run** the program again. What's different this time?

There's an empty function called, `survey_planet()`:

- Write some messages and `print()` them to the console!
- What do you think astronauts will do when they explore new planets?

EXTRA TASKS

- Write your own function which displays a message in the console, then call it at the bottom of the code with the other functions!
- Some of the functions have a word in between the brackets. What do you think this is for?
  ```
  - def navigate_to(destination):
  ```
- What do you think `sleep()` does?

TIPS

Make sure you align your code correctly!

- Code inside of functions should be *indented* to the right (using **TAB**)

Congratulations! You can now create and run your very own *functions*!

PART 3

MAKING DECISIONS USING THE "IF" STATEMENT

We can use code to look at the values of variables, and then tell the computer to make decisions based on them.

To do this, we use something called an "if statement".

```
if condition:
    do something
```

The `condition` is what we are checking for, the code underneath it is what happens if the `condition` is `True`.

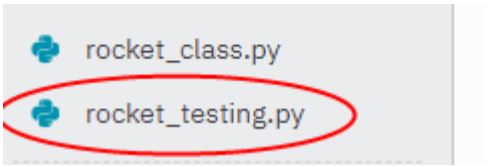
SCENARIO

Your team is in charge of testing rockets to check if they are able to fly. We want to check for a few important things:

- *If* the rocket has an engine
- *If* the rocket has a parachute
- *If* there are any rats in the cockpit
- *If* there are enough astronauts onboard the rocket
- *If* there is enough fuel in the rocket

Notice how all of these *conditions* start with the word **if**?

Switch to "rocket_testing.py".



Have a look at the following code:

```
if has_engine == False:
    print("The rocket doesn't have an engine!")
```

Here, we are checking **if** the rocket **does not** have an engine.

- We are checking the variable has_engine
  - `has_engine` can either be **True** or **False**
- We use the double equals (==) operator to check if `has_engine` is equal to **False** (e.g. the rocket does not have an engine)

MORE ON OPERATORS

We use "*operator*" symbols to compare data within if statements.

As well as the **==** operator, there are a few others:

- **!=** checks if the left side is **not equal** to the right side
- **>** checks if the left side is **greater than** the right side
- **<** checks if the left side is **less than** the right side
- **>=** checks if the left side is **greater than or equal to** the right side
- **<=** checks if the left side is **less than or equal to** the right side

NOW IT'S YOUR TURN!

Write some more if statements to check for the rest of the conditions:

- *If* the rocket **does not** have a parachute
  - Use the `has_parachute` variable
- *If* there **are** any rats in the cockpit
  - Use the `rats_in_cockpit` variable
- *If* there is **enough** fuel in the rocket (***greater than*** *90*)
  - Use the `fuel_level` variable

For each condition, *print* out a useful warning message!

TIPS

- Make sure your code is aligned correctly. It should be in line with the other if statements

- Remember the colon (**:**) at the end of your conditions
- Make sure the `print()` is *indented* one space to the right
    - Use the **TAB** key to indent a line
- When using numbers, you can type them without quotations

And now you know all about *if statements*, *conditions*, and *operators*!

PART 4

LET'S PLAY A GAME

With Python, you can create your very own games!

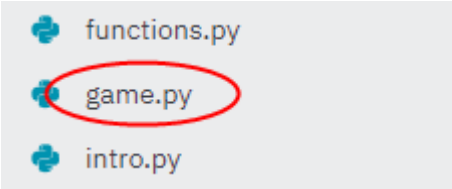
THE MISSION

GU Rocketry has hired your team to make our landing process as efficient as possible.

You have been given a landing simulation program, which will simulate a rocket performing a landing burn in order to safely touch down on a new planet's surface.

THE TASK

Open up "game.py".



Marked at the top of the program, there are three variables:

- `delay`
- `duration`
- `engine_power`

It's your job to find a balance between these three values to land the rocket safely on the planet's surface.

- `delay` is how long the rocket will wait until performing the landing burn
- `duration` is how long the landing burn will last for
- `engine_power` is how strong the burn is

You'll also get more points for using less fuel and landing as fast as possible!

EXTRA TASKS

- If you're feeling adventurous, have a look through the rest of the program code.
- Can you see any variables? What about if statements and functions?

There's so much you can do with coding! We hope you enjoyed this introduction!

Code & Workbook by Timothy Wang for GU Rocketry

GU Rocketry, 2023