The basics

W3Schools is my main resource for making these slides and I **HEAVILY** recommend using it as your go to resource for Python development.

Another resource I would recommend for fixing errors is Stack Overflow

https://www.w3schools.com/python/ https://stackoverflow.com/

Comments

Commenting your code is very important for 3 main reasons.

1. It allows for an outside party like a helper to understand the code.
2. It allows for you to come back to old code and understand it.
3. It makes it easier for other people to contribute to your code.

```
1 # This prints Hello World to the console
2 print("Hello World!")
```

Variables

Adding variables to your code allows for some more advanced functionality and easier readability.

There are 3 main things to do with variables

1. Assign them
2. Use them
3. Change them

VARIABLE NAMES ARE

CASE SENSITIVE!

```
1 message = "Hola!" # Assign
2 print(message) # Use
3 message = "Hello!" # Change
4 print(message) # Use
```

Data Types

There are 4 main different data types in Python

1. Strings - str
2. Boolean - bool
3. Integers - int
4. Float - float

```
1 myVariable = "This is a string!"  # Assign
2 print(type(myVariable))  # <class 'str'>
3 myVariable = False  # Change
4 print(type(myVariable))  # <class 'bool'>
5 myVariable = 1  # Change
6 print(type(myVariable))  # <class 'int'>
7 myVariable = 1.5  # Change
8 print(type(myVariable))  # <class 'float'>
```

Shebang

A shebang allows you to run a file to be ran without any parameters and the OS will know how to execute the file. This only works on Unix system but doesn't do any harm to Windows system so you should get in the habit of using it.

The shebang on the example below allows the OS to realize that the file should be ran using Python 3 if it is not told explicitly.

```
1 #!/usr/bin/env python3
2 print("This can be executed from the command line like this:")
3 print("./04_shebang.py")
```

Arithmetic Operators

There are 7 different arithmetic operators in Python

1. “+” Addition
2. “-” Subtraction
3. “*” Multiplication
4. “/” Division
5. “%” Mod
6. “**” Exponential
7. “//” Floor Division

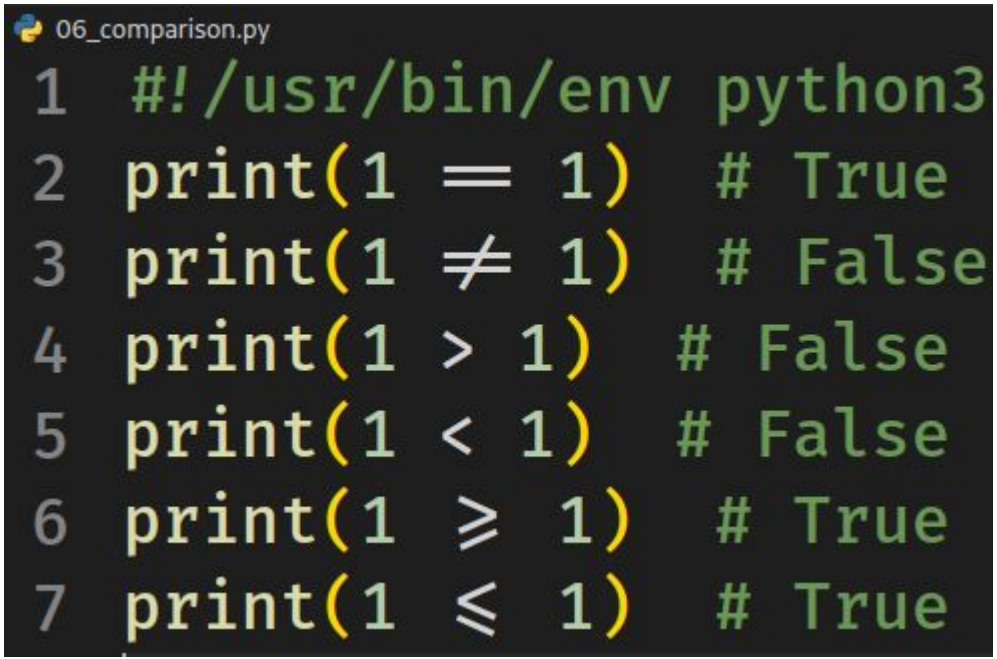
05_operators.py

```
#!/usr/bin/env python3
print(1 + 1.5)  # 2.5
print(1 - 1.5)  # -0.5
print(5 * 5 * 5)  # 125
print(5 / 2)  # 2.5
print(5 % 2)  # 1
print(5 ** 3)  # 125
print(5 // 2)  # 2
```

Comparison Operators

There are 6 different comparison operators in Python

1. “==” Equal
2. “!=” Not equal
3. “>” Greater than
4. “<” Less than
5. “>=” Greater than or equal to
6. “<=” Less than or equal to



If… Else if… Else

If statements are very important when it comes to controlling the flow of your program. When developing your first game, a text based adventure game, you will use a lot of them.

The standard syntax is

if [CONDITION]:

[TAB] Do this

elif:

[TAB] Do this

else:

[TAB] Do this

07_conditionals.py > ...

```
1  #!/usr/bin/env python3
2  friend = True
3  enemy = False
4
5  # Check if friend stranger or foe and react
6  if friend == False and enemy == True:
7      print("Don't acknowledge")
8  elif friend == False and enemy == False:
9      print("Nod down")
10 else:
11     print("Nod up")
```

Loops

Loops are important for things like updating the screen in a game or repeating a task.

There are 2 main types of loops

While Loops and For Loops

While loops run while a condition is true

For loops run until a condition is false

08_loops.py > ...

```
1  #!/usr/bin/env python3
2  number = 0
3  # This will print number
4  # while the number is under 10
5  while number < 10:
6      number = number + 1
7      print(number)
8  # 1 2 3 4 5 6 7 8 9 10
9
10 # This will print num 10 times
11 # Notice how it starts at 0
12 # And ends at 9
13 for num in range(10):
14     print(num)
15 # 0 1 2 3 4 5 6 7 8 9
```

Functions

Functions allow for you to repeat functions that you don’t want to rewrite.

Remember the philosophy DRY

(Don’t Repeat Yourself)

A good example of this is displaying a greeting message.

09_functions.py > ...

```
1  #!/usr/bin/env python3
2
3  def greeting():
4      print("Hello!")
5      print("The date is: 09/15/2022")
6      print("Have a great day!")
7
8  # User opens app
9  greeting()
10
11 # User goes to different menu
12 print("About Us!")
13
14 # User goes back to home menu
15 greeting()
```

Modules

Modules allow you to use other people's code to improve your code and speed up development time.

The syntax for importing a library is:

import [LIBRARY NAME]

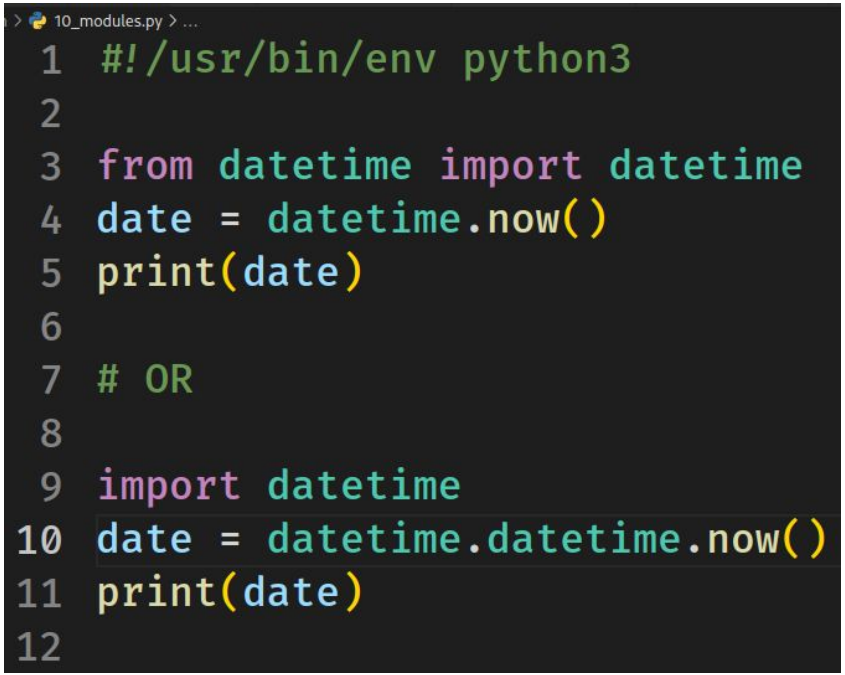
OR

from [LIBRARY] import [FUNCTION]

If it is not a base library you can install it

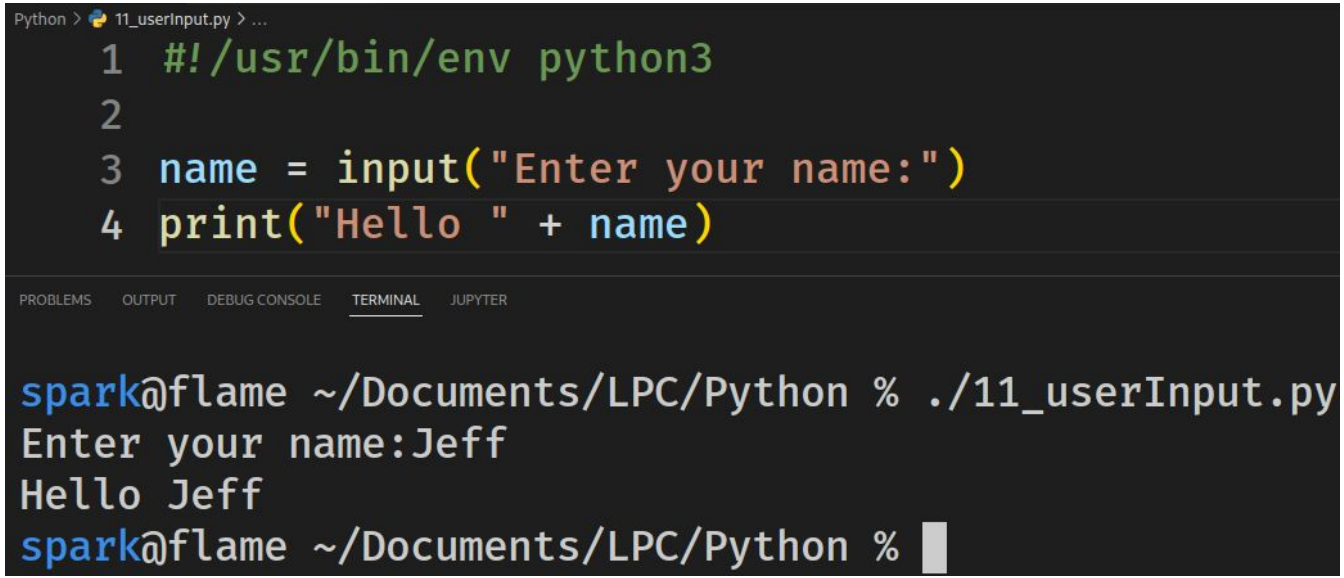
using pip (Package Installer for Python):

pip install --user numPy



User Input

In order to accept input you need to use the input() function. The string inside the input function is what is prompted to the terminal for the user to type in the data that you want to interpret.



Project 1!

We are going to have our first project! It is going to include:

Print statements, Variables, Functions, Modules, User Input, and something else

Your objective is to program a function that when called prints this:

Hello {USER NAME}!
Today is: {TODAY'S DATE}

To get TODAY'S DATE use the datetime module

You will notice an issue in your code, debug it using your resources

Python > Projects > 01_Project.py > ...

```python
#!/usr/bin/env python3

from datetime import datetime

def greeting():
    name = input("What is your name? ")
    date = datetime.now()
    print("Hello " + name + "!")
    print("Today is: " + str(date))

greeting()
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Python Debug Console

```
spark@flame ..ts/Github/LPC/examples/Python/Projects (git)-[main] % ./01_Project.
py
What is your name? Jeff
Hello Jeff!
Today is: 2022-09-09 18:06:27.091451
```

My Solution

Error Handling

Error handling is important to have when dealing with user generated content. For this example we are going throw an error for not having a declared variable and we are going to recover from it

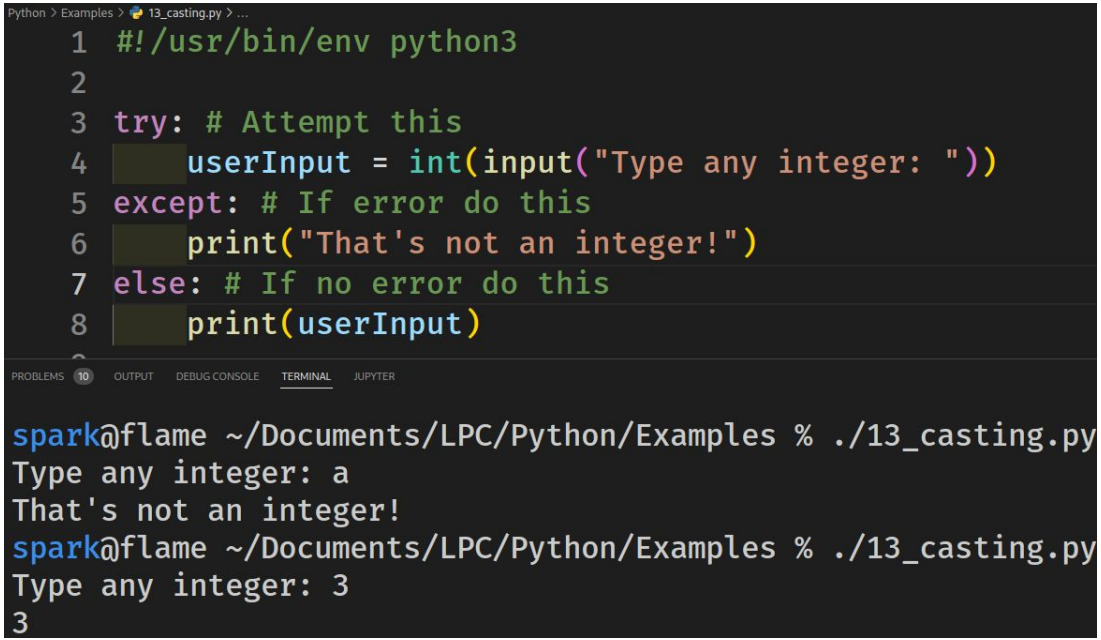
Examples > 12_exceptions.py > ...

```
1  #!/usr/bin/env python3
2
3  # Uncomment the declaration below to have the code run without error
4  # x = 0
5
6  try:  # Run your error prone code in here
7      x = x
8  except:  # Ran if there's an error
9      print("An error occurred!")
10 else:  # Ran if no errors
11     print("No error occurred!")
12 finally:  # Run this no matter what
13     print("This prints no matter what")
```

Casting

Casting is important for converting one data type to another. This can useful when you need to take a user input and make it in to another data type.

Notice how it is wrapped in a try catch statement. This is required in case the user inputs something that can't be cast to an int.



Project 2!

We are going to have our second project! It is going to include:

Print statements, Variables, Arithmetic Operators, If statements, Loops. User Input, Error Handling, and Casting

Your objective is to make a program that will only exit its loop when a number the user gives is divisible by 3 AND 5, otherwise you will have the following inputs:

a - That isn't an integer!

0 - That's not divisible by 3 or 5!

3 - That's only divisible by 3!

5 - That's only divisible by 5!

15 - That's divisible by both 3 and 5!

Challenge - Have line spacing after every guess

```python
#!/usr/bin/env python3

valid = False

while not valid:
    try:
        userInput = int(input("Input an integer that is divisible by 3 and 5: "))
    except:
        print("That isn't an integer!")
    else:
        if userInput % 15 == 0:
            print("That's divisible by both 3 and 5!")
            valid = True
        elif userInput % 5 == 0:
            print("That's only divisible by 5!")
        elif userInput % 3 == 0:
            print("That's only divisible by 3!")
        else:
            print("That's not divisible by 3 or 5!")
    finally:
        print("\n")
```