# PROGRAMMING WITH PYTHON

*By Randal Root*

## Module 08

In this module, you learn to create scripts using custom classes. Classes help you organize functions and data. There are lots you can add to a class to help you with this organization, but here we look at the important ones you should know about, not only in Python but in other languages too.

## Classes

Programs use three basic components: statements, functions, and classes. Here are three common statements: **data** statements, **processing** statements, and **conditional** statements.

```
# Statements
data_int = 123  # Data assignment statement
if data_int == 123:  # Conditional statement
    data_int = 123 + 4  # Processing statement
print(data_int)  # Calling the print() function
```

Listing 1

Statements are organized into **functions** when a programmer wants to use them many times *in a program*. For example, these statements have been "wrapped" in a function called orgranize_statements().

```
# Functions
def organize_statements():
    data_int = 123  # Data assignment statement
    if data_int == 123:  # Conditional statement
        data_int = 123 + 4  # Processing statement
    print(data_int)  # Calling the print() function
```

Listing 2

Functions are organized into **classes** when a programmer believes they will be used many times in many programs.

```
# Classes
class organize_functions:

    @staticmethod
    def organize_statements():
        data_int = 123  # Data assignment statement
        if data_int == 123:  # Conditional statement
            data_int = 123 + 4  # Processing statement
        print(data_int)  # Calling the print() function Listing 3
```

Listing 3

***Note:*** Oddly enough, data and functions in a class have different names. Variables and constants in a class are called either fields, attributes, or properties, while Functions in a class are called Methods.

One program can have many statements, many functions, and many classes. It all depends on how complex the program is. Using functions to organize statements and classes to organize functions makes complex programs easier to create and manage.

Listing 4 shows examples of statements, functions, and classes in a script. One thing to note is that the first block of statements immediately runs when the script starts. However, the function and the class code will only load into memory. Their code is used later in the script when the functions are called!

```
# ------------------------------------------------ #
# Title: Demo01-Statements-Functions-Classes
# Description: A statements, functions, and classes
# ChangeLog: (Who, When, What)
# RRoot,1.1.2020,Created Script
# ------------------------------------------------ #

# Modern programs are usually created using these three components:
```

```
# Statements
data_int = 123  # Data assignment statement
if data_int == 123:  # Conditional statement
    data_int = 123 + 4  # Processing statement
print(data_int)  # Calling the print() function


# Functions
def organize_statements():
    data_int = 123  # Data assignment statement
    if data_int == 123:  # Conditional statement
        data_int = 123 + 4  # Processing statement
    print(data_int)  # Calling the print() function


# Classes
class organize_functions:

    @staticmethod
    def organize_statements():
        data_int = 123  # Data assignment statement
        if data_int == 123:  # Conditional statement
            data_int = 123 + 4  # Processing statement
        print(data_int)  # Calling the print() function


# ---------------------------

input("Press enter to call the function in the main body of the script")
organize_statements()

# Call the function within the class (aka. Method)
input("Press enter to call the class function (aka. Method)")
organize_functions.organize_statements()
```

Listing 4

## Data Classes vs. Processing Classes

Classes are designed to focus on either data, processing, or interaction (input and output). For example, a developer might create one class for processing data to and from a file, naming it something like "FileManager." They also might create another class for managing the customer data, naming it something like "Customer." The focus of the FileManager" class would be to perform a set of actions, while the focus of the "Customer" class would be to organize data about a customer.

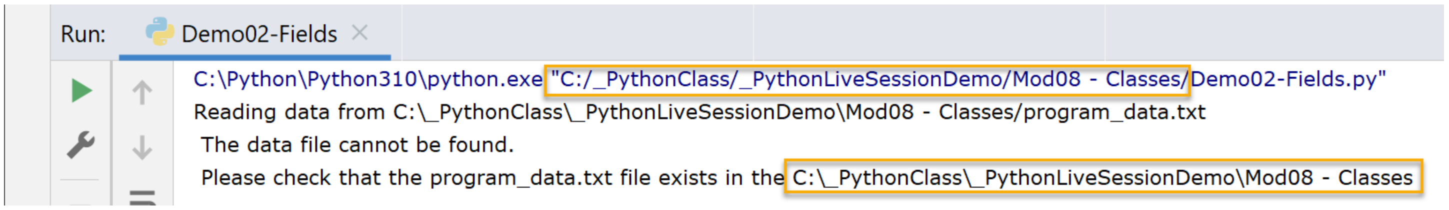
## A Standard Class Pattern

Classes typically have **Fields, Constructors, Properties, and Methods**. Like scripts, class code follows a general design pattern in most of the languages. To learn more about how classes work, let's look at each of these components, starting with fields.

### Fields

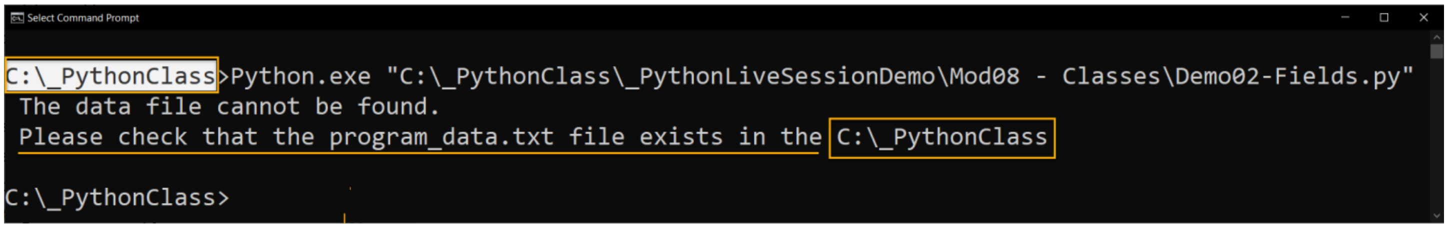
Fields are data members of a class. You create them using variables and constants. Listing 5 shows an example of creating three constant fields describing information about a program's data file. The results of this program are shown in figures one and two.

```
# ------------------------------------------------- #
# Title: Demo02-Fields
# Description: A simple example of a class field
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

#--- Make the class ---
class FileInfo:
    # --Fields--
    FILE_FOLDER = os.getcwd()  # Python's get the current working directory function
    FILE_NAME = 'program_data.txt'
    FULL_PATH = FILE_FOLDER + '/' + FILE_NAME# End of class


# --- Use the class ----
try:
    file_obj = open(FileInfo.FULL_PATH, 'r')
except FileNotFoundError as e:
    error_msg = 'Please check that the ' + FileInfo.FILE_NAME
    error_msg += 'file exists in the  ' + FileInfo.FILE_FOLDER
    print(error_msg)
```
Listing 5



```
C:\Python\Python310\python.exe "C:/_PythonClass/_PythonLiveSessionDemo/Mod08 - Classes/Demo02-Fields.py"
Reading data from C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes/program_data.txt
 The data file cannot be found.
 Please check that the program_data.txt file exists in the C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes
```

Figure 1: The results of listing 5 in a PyCharm window



```
C:\_PythonClass>Python.exe "C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes\Demo02-Fields.py"
 The data file cannot be found.
 Please check that the program_data.txt file exists in the C:\_PythonClass

C:\_PythonClass>
```

Figure 2: The results of listing 5 in a terminal window

**Important:** Remember that the current working directory in PyCharm and in the Command window may be different since the current working directory is based on the folder your program is using when its running!

## LAB 8-1

In this lab, you create a simple class to organize data about a file.

1. **Use** the code in Listing 5 to create the FileInfo class. Try typing the code for practice first, but you can copy and paste if you have problems.

2. **Test** the code in PyCharm and in a terminal window.

3. **Note** the difference between the current working directory's location. The figure above shows an example from my computer, but yours may have a different path.

## Objects vs. Classes

When the class's code loads into memory, you either use the class's code directly or indirectly by making a copy of the class's code. To use the code indirectly you create an "object instance." An object instance (or just object) can be thought of as an independent copy of the classes code.

For example, let's say we have a customer class with two fields (variables) called Id and Name.

```
class Customer:
 Id
 Name
```

To **directly** use the Customer's fields, **use code like this**:

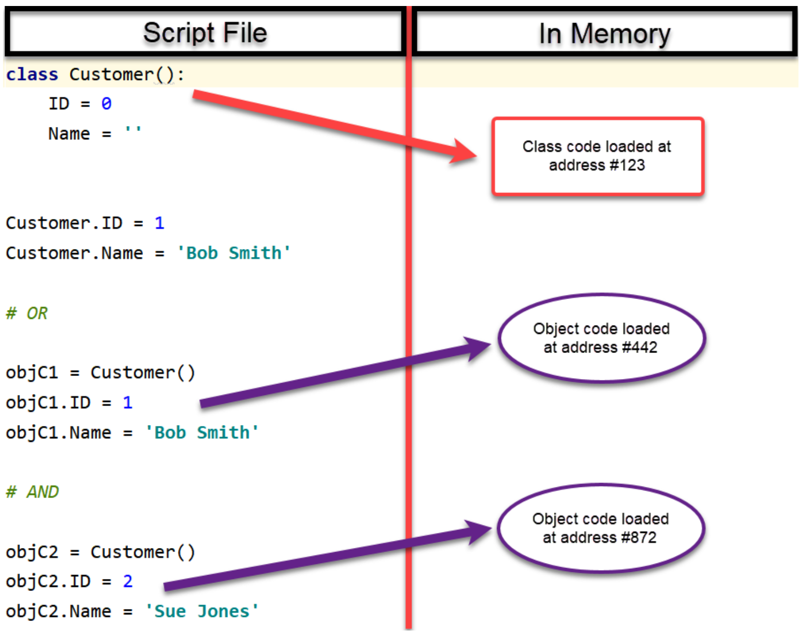
```
Customer.Id = 100
Customer.Name = "Bob Smith"


Customer.Id = 200
Customer.Name = "Sue Jones"
```

The problem with this approch is that changing the id or name overwrites the previous data. However, you can use an object instance of the class to avoid this.

You create and use an object instance a class like this:

```
objC = Customer()
objC.Id = 100
objC.Name = "Bob Smith"
```

One advantage of using a copy of the code is that you can have multiple object instances, each with a different address in memory. The data in each object instance is separate from all other objects. This is useful since each object can hold different data for each customer.


Script File
In Memory
Class code loaded at address #123
Object code loaded at address #442
Object code loaded at address #872

```
class Customer():
    ID = 0
    Name = ''


Customer.ID = 1
Customer.Name = 'Bob Smith'

# OR

objC1 = Customer()
objC1.ID = 1
objC1.Name = 'Bob Smith'

# AND

objC2 = Customer()
objC2.ID = 2
objC2.Name = 'Sue Jones'
```

Figure: A simple example of objects made from a customer class

In general, use a class directly if it does not unique data or its focus is on processing data, and use object instances of the class if its focus is on storing unique data (that way each copy has its own data). This generalization may not always be true, but it is enough to provide a good starting point.

***Notes**:*

- *The address numbers are just for illustration. I am using **made-up address numbers** to indicate different locations in a computer's memory.*
- *The **concepts** in this module are advanced and may **take time to understand**. Be patient; they become clearer as you work with class and functions!*
- *This basic explanation of the subject is purposely simplified to help students get a general idea of the topic. **More technical explanations are covered in higher level courses**.*

## Constructors

When you create an object instance from a class, you call the class's name as if it were a function. Using the class's name like a function automatically calls the class's constructor method.

Constructors are special methods (functions) that automatically run when you create an object from a class. Constructors set initial data values, which is why they are often called an "**initializer**." In fact, Python's constructors are always named "**__init__**".

Constructor methods use parameters to capture the initial values as shown in listing x.

```
# ------------------------------------------------- #
# Title: Demo03-Constructors
# Description: A class with a constructor
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

class Person():
    # --Fields--
    first_name_str = ""

    # -- Constructor --
    def __init__(self, first_name=''):  # The default is an empty string
        self.first_name_str = first_name

# --End of class--

# --- Use the class ----
objP1 = Person()  # With no arguments
objP1.first_name_str = "Bob"

objP2 = Person(first_name="Sue")  # With one parameter and argument

print(objP1.first_name_str)  # Will be empty because there was no argument
print("-------------")
print(objP2.first_name_str)  # Will have first name of Sue
```
Listing 6

***Note:** constructors are a specialized function, so you use them as a function by passing arguments into the parameters. However, remember, they only run once, only when a new object instance of a class is created!*

## The Self Keyword

You probably noticed the use of the keyword **"self"** in the constructor method. This keyword is used to refer to data or functions found in an object instance, and not directly in the class.

To understand the "self" keyword, remember that the class’s code immediately loads into memory when your script starts running and then waits to be used, either directly or through an object instance.

The following code first loads the Person class into memory and then an creates object objP1 and objP2 from the Person class.

```
class Person():  # 1. Load this class into memory
    # --Fields--
    first_name_str = ""

    # -- Constructor --
    def __init__(self, first_name=''):
        self.first_name_str = first_name


objP1 = Person("Bob")  # 2. Use the class
objP2 = Person("Sue")  # 2. Use the class
```

While the class's code only loads into memory once, it can have many object instances of a class, each representing a "copy" of the classes code! In Python, you identify which copy is referenced using the pronoun "self." Just as two people conversing might each refer to themself!

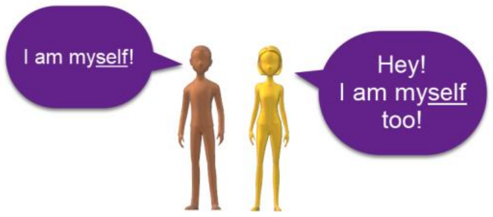


Figure 2: Using the "myself' pronoun is contextual

Other languages use the word "this" instead, but in Python it's always "self." The people who made the Python language made a rule to include a parameter called "self" in each method meant to be used from an object instance.

**Notes:**

- Oddly, you **cannot not pass an argument to the "self" parameter**, it is automatically filed with the object's referenced location in memory.
- When a **method is used directly** from the class, you **leave out the "self" parameter and** mark the method with the **@staticmethod** decorator as we have seen in earlier modules.

## LAB 8-2

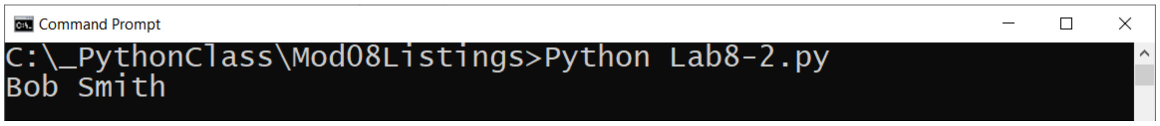
In this lab, you **add** a **constructor** to the Person class you made in Lab 8-1.

1. **Modify** the Person class to include a constructor using the code in Listing 6. Try typing the code for practice first, but you can copy and paste if you have problems.

2. **Add** a last_name parameter in addition to the first_name parameter to the class's constructor.

3. **Use** these parameters to set a first_name_str and last_name_str fields.

4. **Test** the code by creating an object instance, setting the values of the fields, then printing the values of the fields.



```
C:\_PythonClass\Mod08Listings>Python Lab8-2.py
Bob Smith
```

Figure: The results of Lab 8-2

**Important:** We covered Fields because it is a common option in programming languages. However, **Python does not use them this way.** Instead, we will use Attributes to hold the data in our object instances.

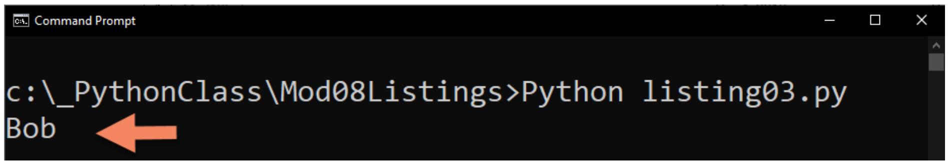
## Attributes

Instead of standard fields**, Python uses "virtual" fields** to hold internal object data. These virtual fields, **called Attributes,** can be created by declaring the variable in the constructor as show here:

```
# -------------------------------------------------- #
# Title: Demo04-Attributes.py
# Description: A class with an attribute
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# -------------------------------------------------- #

class Person(object):
    # --Fields--
    # first_name_str = ""  <- Delete this. Python does not use fields for instance data

    # -- Constructor --
    def __init__(self, first_name):
        # Attributes
        self.first_name_str = first_name  # this is a attribute (virtual field)

# --End of class--

# --- Use the class ----
objP1 = Person("Bob")
print(objP1.first_name_str)  # Using the empty explicit field
```

Listing 7

When the code runs, it still acts as if the field existed, but it is now using an attribute or "virtual field."



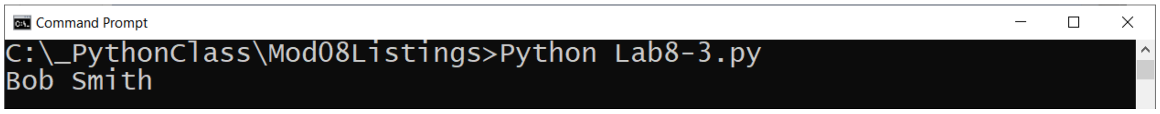
```
c:\_PythonClass\Mod08Listings>Python listing03.py
Bob
```

Figure 4: The results of Listing 7.

## LAB 8-3

In this lab, you modify the constructor of the Person class you made in Lab 8-2 to use **attributes** instead of fields.

1. **Delete** the first_name_str and last_name_str fields since we are no longer using them.
2. **Verify** that the constructor **sets** the first_name_str and last_name_str attributes with parameter data as it did before. You do not need to make any other changes.
3. **Test** the code still works by creating an object instance, setting the values of the attributes, and printing the attribute values.



```
C:\_PythonClass\Mod08Listings>Python Lab8-3.py
Bob Smith
```

Figure 5: The results of Lab 8-3.

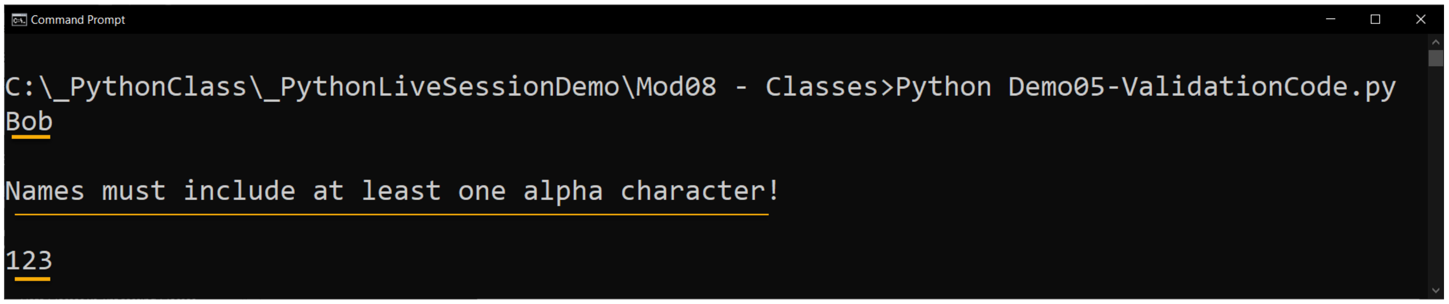
**Note:** Try typing the code for practice first, but you can copy and paste if you have problems.

## Adding Error Handling

Whenever you work with data there is a chance that a human can enter incorrect values. As such, it is important to add validation code to keep your data "clean." For example, we can add validation code to the constructor to stop a string of number being used for a name using Python's isnumeric() function. I am also adding formatting using the title() function to make the data more consistent.

```
# ------------------------------------------------- #
# Title: Demo05- Validation Code
# Description: A class with an attribute
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

class Person(object):

    # -- Constructor --
    def __init__(self, first_name):
        # Attributes
        self.first_name_str = '' # declare the attributes first
        try:
            if str(first_name).isnumeric():  # test that parameter data is valid
                raise ValueError
            else:
                self.first_name_str = first_name.title()  # set and format attribute
        except ValueError as v:
            print(v)
            print('Names must include at least one alpha character!')

# --End of class--

# --- Use the class ----
objP1 = Person("bob")
print(objP1.first_name_str)
objP1 = Person("123")
print(objP1.first_name_str)

# However, it does not stop us from using numbers later(an issue we will fix shortly!)
objP1.first_name_str = '123'
print(objP1.first_name_str)
```

Listing 8. Adding validation code



```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo05-ValidationCode.py
Bob

Names must include at least one alpha character!

123
```

Figure 6: the results of listing 8.

## Private Attributes

It is  a best practice to mark an attribute as private whenever its values are controlled by a method, like our constructor.

```
def __init__(self, first_name):
    self.__first_name = first_name
```

Note, I have added (2) underscores before the attribute's name. This marks the attribute as "*private*" and indicates that developers should not access the attribute from code outside of the class. Instead, they should use a method to work with the attribute's values.

Oddly, Python does not vigorously enforce this privacy, like most other languages. So, it is up to you and your coworkers to respect the privacy of an attribute. The recommended way of doing this is by using the property functions to get and set attribute data indirectly.

## Properties

Properties are special functions used to manage attribute data. You typically create two properties for each attribute, one for "getting" data and one for "setting" data. In fact, these types of function are often called "**Getters**" and "**Setters**" or "Accessors" and "Mutators."

Getter property functions let you access the data and optionally add formatting code. Python uses the @property decorator to indicate a fuction is a "getter" like this:

```
@property  # (getter or accessor)
def first_name(self):  # This is the name used to work with the data
    return str(self.__first_name_str).title()  # Formatting data in title case
```

Setter property functions let you add code for both validation and error handling. If a value passed into the Properties parameter is valid, then it is assigned to the attribute. You create a setter like any other function, but it must include the @name_of_property.setter decorator**.**

```
@first_name.setter  # (setter or mutator)
def first_name(self, value):  # make name match the getter!
    if str(value).isnumeric() == False:
        self.__first_name_str = value
    else:
        raise Exception("Names cannot be numbers")
```

**Note**, that the **property** is now responsible for **managing** the double underscore (private) **attribute**!

**Important:** The **Setter decorator and function name must match the name of the Getter** for them to be a "getter and setter" pair! This odd and inconsistent syntax is unique to Python.

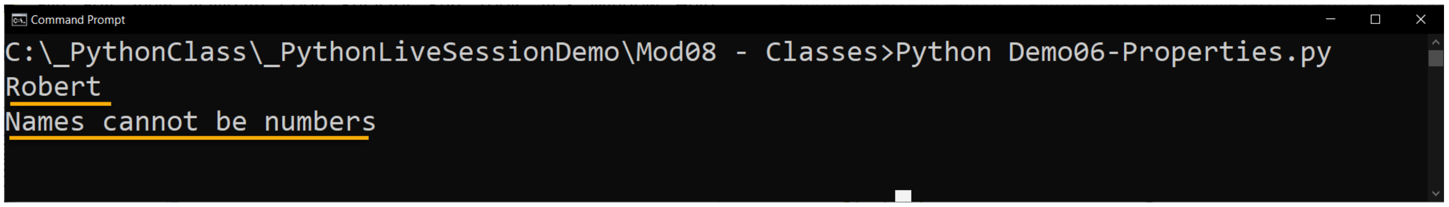
Next, you **modify the constructor to use the properties instead of the attributes**. This change forces you code to use the validation logic within those properties.

```
# -- Constructor --
def __init__(self, first_name):
    # Attributes
    # self.__first_name_str = first_name  # Don't use attributes in the constructor!

    self.first_name = first_name  # Use Properties instead!
```

Here is a listing of our updated code:

```python
# ------------------------------------------------- #
# Title: Listing06 - Properties
# Description: A class with an attribute
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

# --- Make the class ---
class Person:
    # --Fields--
    #strFirstName = ""

    # -- Constructor --
    def __init__(self, first_name):
        # Use a property to set the attribute
        self.first_name = first_name
        # The property and parameter can have the same name due to the self context

    # -- Properties --
    @property  # You don't use for the getter's directive!
    def first_name(self):  # (getter or accessor)
        return str(self.__first_name).title()  # Format attribute as Title case

    @first_name.setter  # The @NAME.setter must match the getter's name!
    def first_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__first_name = value
        else:
            raise Exception("Names cannot be numbers")

# --End of class--

# --- Use the class ----
objP1 = Person("Bob")

objP1.first_name = 'robert'  # using the Setter property
print(objP1.first_name)  # using the Getter property

try:
    objP1.first_name = '123'  # testing that a number causes a validation ERROR
except Exception as e:
    print(e)
```

Listing 9



```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo06-Properties.py
Robert
Names cannot be numbers
```

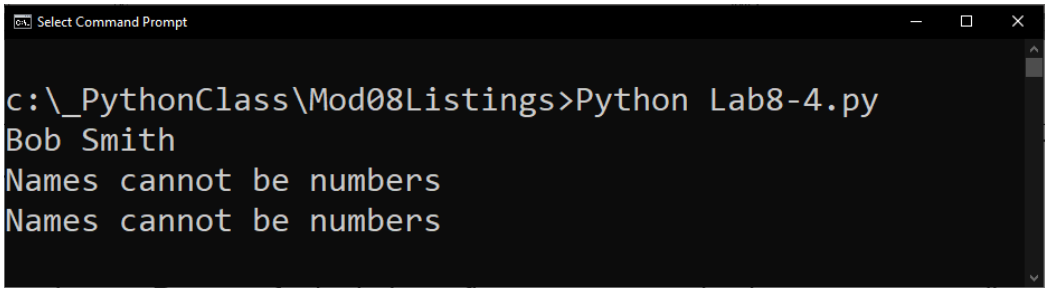
Figure 7: The results of listing 9.

**Important:** **Don't use fields or attributes in Python constructors.** If we do, the property's validation logic won't be used when the object is created.

## LAB 8-4

In this lab, you **modify** the Person class you made in Lab 8-3 to use a getter and setter property function for the first and last name **attributes**. You will also make the attributes private then add validation or formatting code to each property function.

1. **Create** a getter and setter Property function called first_name to manage the private __first_name attribute. Make sure to use formatting and validation as shown in listing 9.
2. **Create** a getter and setter Property function called last_name to manage the private __last_name attribute. Make sure to use formatting and validation as shown in listing 9.
3. **Modify** the constructor to use the property names **self.first_name** and **self.last_name**, instead of the attribute names.
4. **Test** the code by creating an object instance, setting the properties, then printing the values of the first and last name properties.
5. **Try** entering a number for the first and last name to **test** that an error occurs.



```
c:\_PythonClass\Mod08Listings>Python Lab8-4.py
Bob Smith
Names cannot be numbers
Names cannot be numbers
```

Figure 8: The results of Lab 8-4.

**Note:** Try typing the code for practice first, but you can copy and paste the code from listing 9 if you have problems.

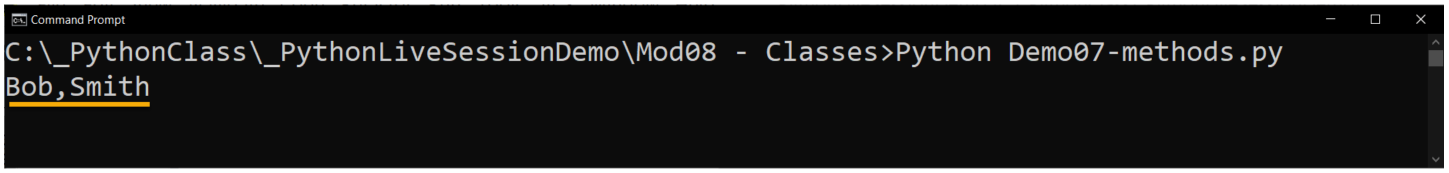
## Methods

While functions that manage attribute data are called properties, other functions inside of a class are called Methods. Methods are just like normal functions, letting you organize statements into named groups, but are part of a class's code instead of being just part of a script's general code.

Most classes include a method that returns some or all the class's data as a string. Here is an example of how that might work using the first and last name in a comma separated value format (which is commonly used throughout the IT industry.)

```
# ------------------------------------------------- #
# Title: Demo-07-Methods
# Description: A class methods
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

# --- Make the class ---
class Person:

    # -- Constructor --
    def __init__(self, first_name, last_name):
        # Use a property to set the attribute
        self.first_name = first_name
        self.last_name = last_name

    # -- Properties --
    @property  # You don't use for the getter's directive!
    def first_name(self):  # (getter or accessor)
        return str(self.__first_name).title()  # Format attribute as Title case

    @first_name.setter  # The @NAME.setter must match the getter's name!
    def first_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__first_name = value
        else:
            raise Exception("Names cannot be numbers")

    @property  # You don't use for the getter's directive!
    def last_name(self):  # (getter or accessor)
        return str(self.__last_name).title()  # Format attribute as Title case

    @last_name.setter  # The @NAME.setter must match the getter's name!
    def last_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__last_name = value
        else:
            raise Exception("Names cannot be numbers")

    def to_string(self):
        """  Returns object data in a comma separated string of values

        :return: (string) CSV data
        """
        object_data_csv = self.first_name + ',' + self.last_name
        return object_data_csv

# --End of class--

# --- Use the class ----
objP1 = Person("bob", "smith")
print(objP1.to_string())
```

Listing 10



```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo07-methods.py
Bob,Smith
```

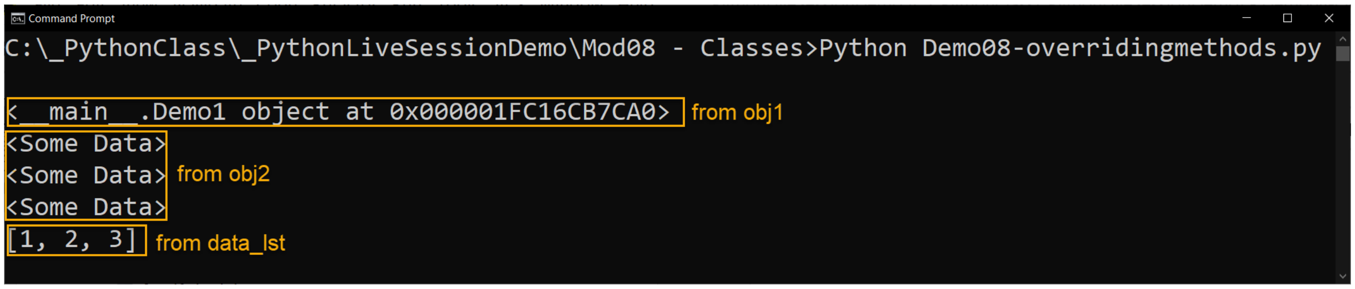
Figure 9: The results of Listing 10.

## The Built-in "__str__()" Method

Python has a built-in method called the **"__str__()**" method for returning the class's data as a string. This method exists in every class, even if you do not add it to the class yourself!

Python's invisible "__str__()" method defaults to returning the memory address of the class's object instance (which you may have seen this going through the course.) Here is an example:

```
# ---------------------------------------------------------------- #
# Title: Demo-08-Overriding the built-in __str __() method
# Description: A overriding the __str__() method
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ---------------------------------------------------------------- #

class Demo1:
    var1 = "Some Data"

obj1 = Demo1()  # This object uses the default __str__() method
print(obj1.__str__())

class Demo2:
    var1 = "Some Data"
    def __str__(self):
        return '<' + self.var1 + '>'

obj2 = Demo2()  #  object uses the overridden __str__() method
s = str(obj2)
print(str(obj2))  # str() automatically calls the  __str__() method
print(obj2)  # print() function automatically calls the  __str__() method
print(obj2.__str__())  # but you can also call the __str__() method directly

# Python's List class overrides the __str__() method too, but it is not very useful.
data_lst = [1,2,3]
print(data_lst)
```
Listing 11



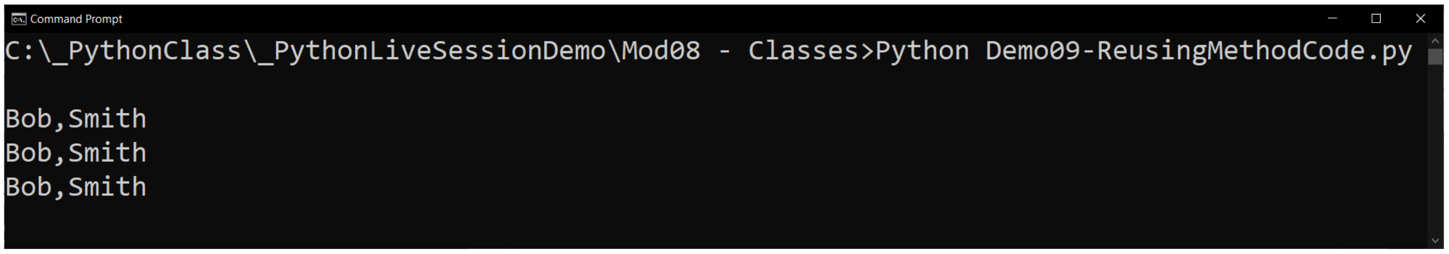
```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo08-overridingmethods.py

<__main__.Demo1 object at 0x000001FC16CB7CA0>
<Some Data>
<Some Data>
<Some Data>
[1, 2, 3]
```

Figure 10: The results of listing 11.

In our Person example, we can override the __str__() method to use the same code as our to_string() method, by calling our method from the Python built-in method like this:

```
# ---------------------------------------------------- #
# Title: Demo-09-Reusing method code
# Description: A class methods
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ---------------------------------------------------- #

# --- Make the class ---
class Person:
    # -- Constructor --
    def __init__(self, first_name, last_name):
        # Use a property to set the attribute
        self.first_name = first_name
        self.last_name = last_name

    # -- Properties --
    @property  # You don't use for the getter's directive!
    def first_name(self):  # (getter or accessor)
```

```
        return str(self.__first_name).title()  # Format attribute as Title case

    @first_name.setter  # The @NAME.setter must match the getter's name!
    def first_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__first_name = value
        else:
            raise Exception("Names cannot be numbers")

    @property  # You don't use for the getter's directive!
    def last_name(self):  # (getter or accessor)
        return str(self.__last_name).title()  # Format attribute as Title case

    @last_name.setter  # The @NAME.setter must match the getter's name!
    def last_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__last_name = value
        else:
            raise Exception("Names cannot be numbers")

    def to_string(self):
        """  Returns object data in a comma separated string of values

        :return: (string) CSV data
        """
        object_data_csv = self.first_name + ',' + self.last_name
        return object_data_csv

    def __str__(self):
        """  Overrides Python's built-in method to
             return object data in a comma separated string of values

        :return: (string) CSV data
        """
        return self.to_string()
# --End of class--

# --- Use the class ----
objP1 = Person("bob", "smith")
print(objP1.to_string())
print(objP1.__str__())
print(objP1)
```
Listing 11



```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo09-ReusingMethodCode.py

Bob,Smith
Bob,Smith
Bob,Smith
```

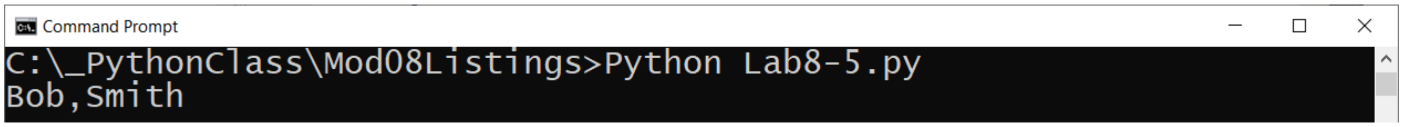
Figure 11: The result of listing 11.

***Note:*** *There are many automatic and invisible built-in methods included in every class. We will talk more about them and how they work module 9.*

## LAB 8-5

In this lab, you **modify** the Person class you made in Lab 8-4 to included a to_string() method and an overridden "__str__()" method.

1. **Add** a new method to named to_string() that returns both the first_name and last_name with a comma separator.
2. **Override** the **"**__str__()" built-in method to use your to_string() method.
3. **Test** the code by creating an object instance, setting the properties, then using print() to run the automatically run the object's "__str__()" method.



Command Prompt

```
C:\_PythonClass\Mod08Listings>Python Lab8-5.py
Bob,Smith
```

Figure 12: The results of Lab 8-5

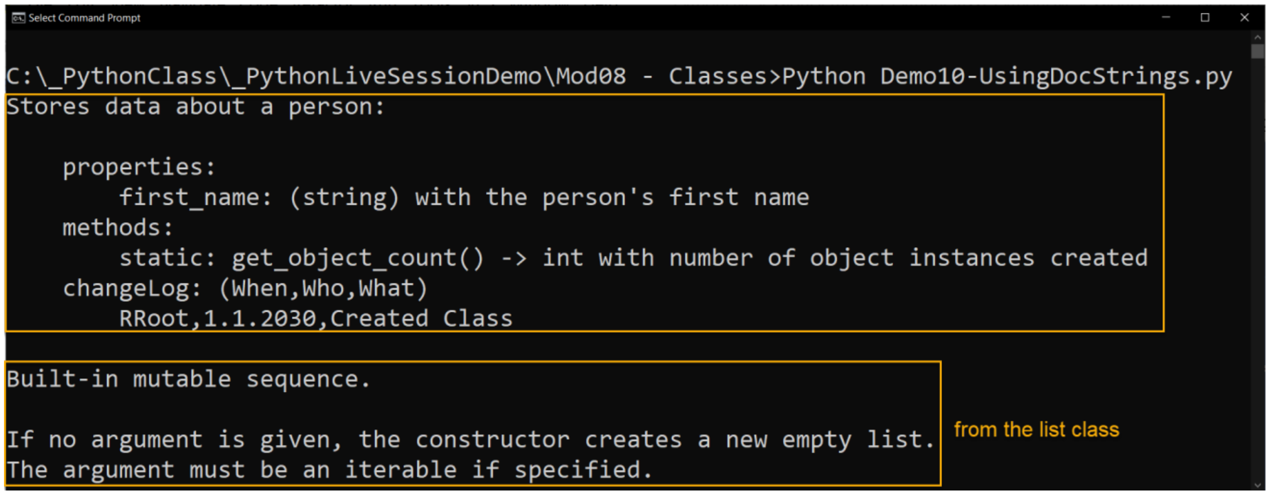
**Note:** Try typing the code for practice first, but you can copy and paste the code from listing 11 if you have problems.

## DocStrings

Just as we did with functions, we should include a docstring for our classes. It can be helpful if developers include additional notes in a docstring. When they do, Integrated development environments like PyCharm can display tooltips showing the developer's notes (use ctrl + q to activate this option in PyCharm). You can also show the "DocString" using the built-in and inherited __doc__ property (Figure 16).

```
# ------------------------------------------------- #
# Title: Demo-10 Class Docstring
# Description: A class with a docstring
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# ------------------------------------------------- #

class Person:
    """Stores data about a person:

    properties:
        first_name: (string) with the person's first name
    methods:
        static: get_object_count() -> int with number of object instances created
    changeLog: (When,Who,What)
        RRoot,1.1.2030,Created Class
    """
    # pass is used as a temporary placeholder for code that will be added
    pass  # TODO: Add code to the class

# --- Use the class ----


print(Person.__doc__)   # You don't use () for this one, because it actually a field
print(list.__doc__)   # Here is one from the Python programmers.
```

Listing 12

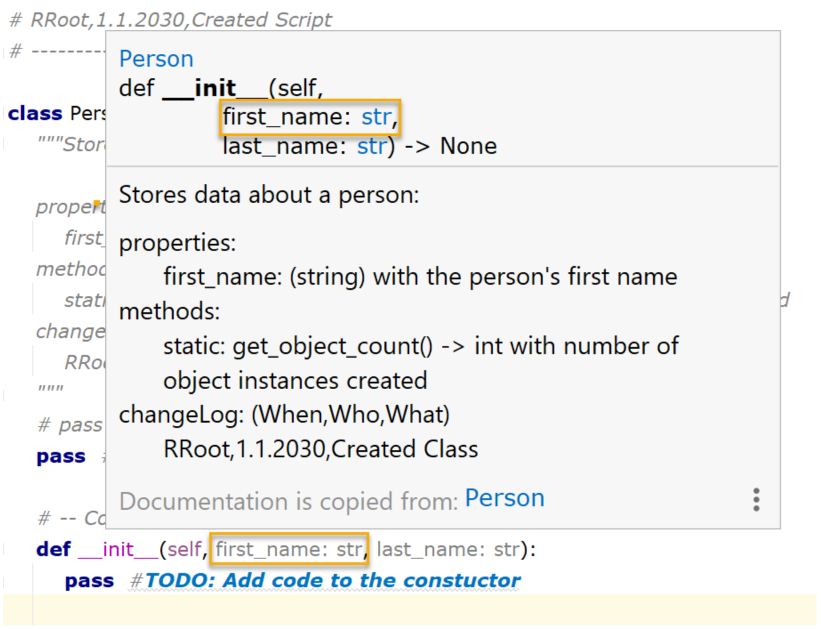


```
C:\_PythonClass\_PythonLiveSessionDemo\Mod08 - Classes>Python Demo10-UsingDocStrings.py
Stores data about a person:

    properties:
        first_name: (string) with the person's first name
    methods:
        static: get_object_count() -> int with number of object instances created
    changeLog: (When,Who,What)
        RRoot,1.1.2030,Created Class

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

Figure 13: The results of listing 12.

## Type Hints

Another documentation feature recently added to Python is "type hints." These identify what type of data is expected in a parameter. Type hints can help other developers understand what is wanted and some IDEs provided tips to reduce mistyping.



```
# RRoot,1.1.2030,Created Script
# --------
class Pers
    """Stor
    propert
        first
    methoc
        stati
    change
        RRo
    """
    # pass
    pass

    # -- Co
    def __init__(self, first_name: str, last_name: str):
        pass  #TODO: Add code to the constuctor
```

```
Person
def __init__(self,
             first_name: str,
             last_name: str) -> None

Stores data about a person:

properties:
    first_name: (string) with the person's first name
methods:
    static: get_object_count() -> int with number of
    object instances created
changeLog: (When,Who,What)
    RRoot,1.1.2030,Created Class

Documentation is copied from: Person
```

Figure 14: Type Hints in PyCharm IDE

**Note:** Unlike other languages, the Python runtime does not yet enforce parameter types, so incorrect argument types can still be passed into to a parameter.

## GitHub Desktop

In this course, we've used GitHub's website through a web browser, but often developers work with GitHub differently. Instead of browser, they work with GitHub on their local computers using either with a command prompt application or a desktop application. In both cases, the communication between the GitHub website and the local computer is handled by a program called **"Git.**"

### GIT

The Git software manages versions of one or more files. It allows you to make a clone (copy) of a file, then make changes to clone, and save it as a new version of the same file. All while maintaining a copy of the original version. In addition to managing the cloned file on your computer, by default, Git uses the GitHub website to store backup files in the "Cloud."

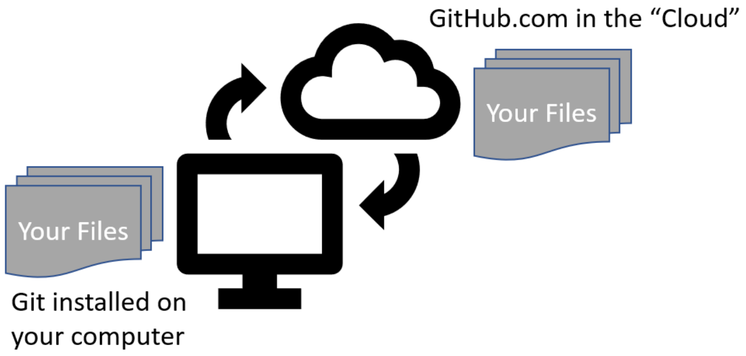
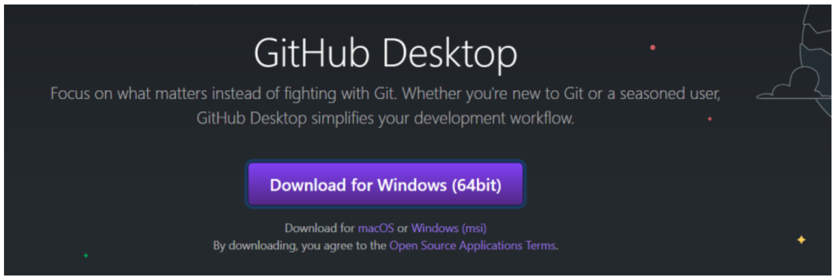


Figure 15. Git and GitHub work together to manage files

## GitHub Desktop

In most organizations, developers use command shell to interact with the website from their local computers, but in this module, we start with something more visual; GitHub Desktop. GitHub Desktop is a free application you can install on both Windows and Mac OS.

To install GitHub Desktop, you navigate to its download page (**https://desktop.github.com/**), select your operating system, and download the installation file. Then start the installation, which is quick and straightforward!



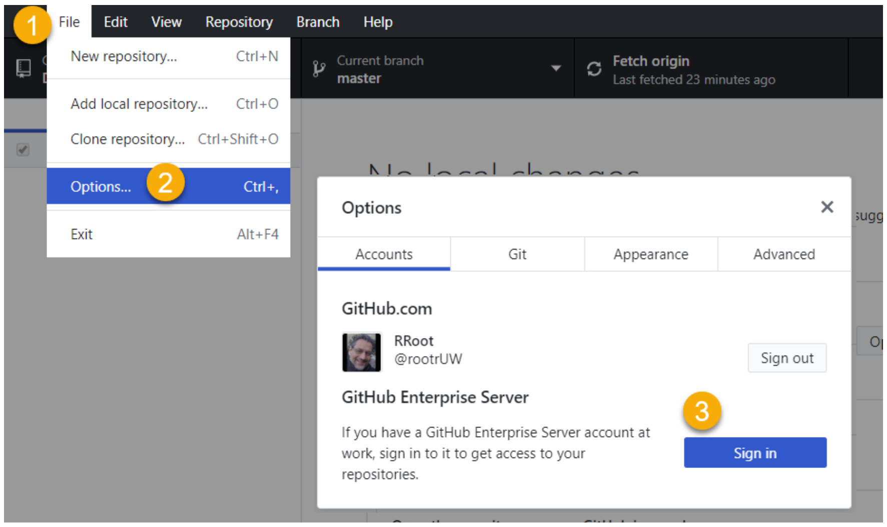
GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Download for macOS or Windows (msi)
By downloading, you agree to the Open Source Applications Terms.

Figure 16. The GitHub Desktop download page

Once it installs, it opens the **application and asks you to log in to your GitHub website account**. You can always change that account later using the File -> Options menu, then the Sign in and Sign buttons (Figure 17).



If you have a GitHub Enterprise Server account at work, sign in to it to get access to your repositories.

Figure 17. Changing the GitHub account option

Once you are signed in, you can see a listing of your GitHub repositories on the right-hand side of the UI. Click on a repository, then **click the "Clone" button to download a copy of your repository's files** (Figure 18). This button launches the "Clone a repository" dialog box.

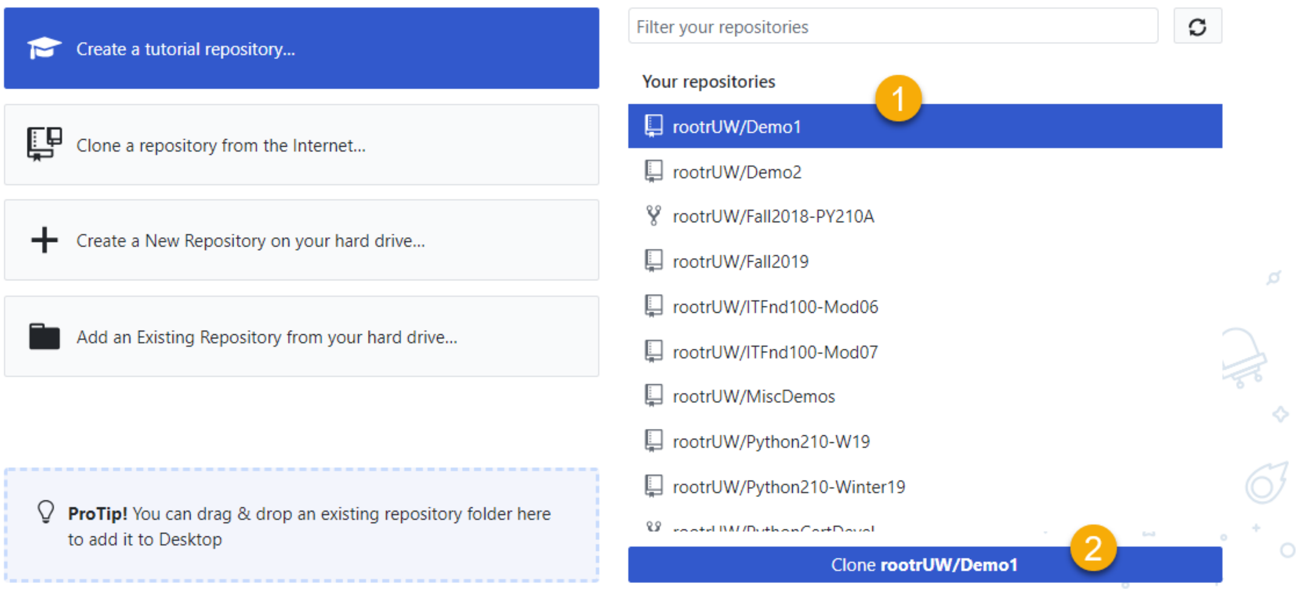
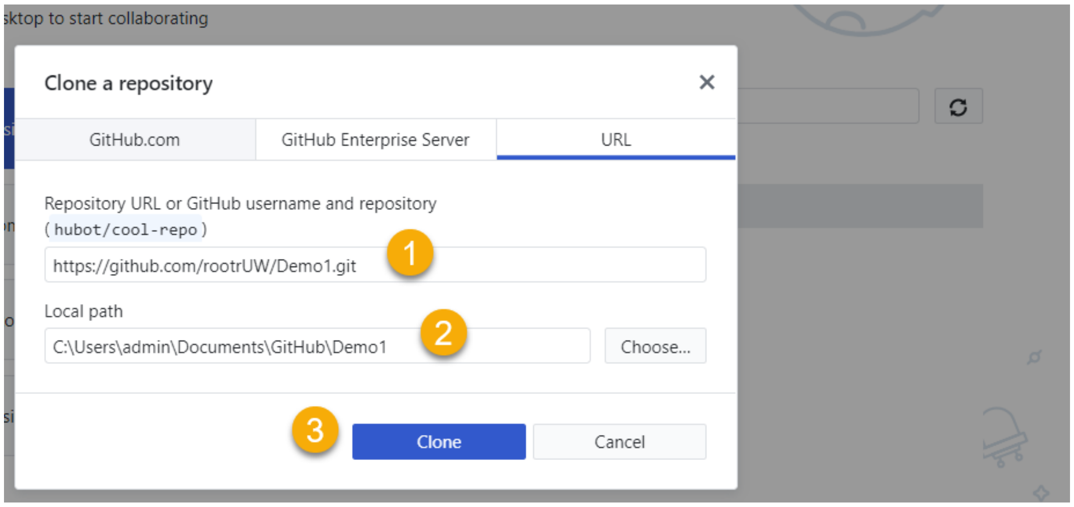

Create a tutorial repository...
Clone a repository from the Internet...
Create a New Repository on your hard drive...
Add an Existing Repository from your hard drive...
ProTip! You can drag & drop an existing repository folder here to add it to Desktop
Filter your repositories
Your repositories
1
rootrUW/Demo1
rootrUW/Demo2
rootrUW/Fall2018-PY210A
rootrUW/Fall2019
rootrUW/ITFnd100-Mod06
rootrUW/ITFnd100-Mod07
rootrUW/MiscDemos
rootrUW/Python210-W19
rootrUW/Python210-Winter19
2
Clone rootrUW/Demo1

Figure 18. The repository listing

In the "Clone a repository" dialog box, **verify** the GitHub.com **repository and the local folder** where the files are copied to, **before clicking the "Clone" button** (Figure 19). This button starts the download process.



Repository URL or GitHub username and repository (`hubot/cool-repo`)

https://github.com/rootrUW/Demo1.git

Local path

C:\Users\admin\Documents\GitHub\Demo1

Figure 19. The "Clone a repository" dialog box

Once the files download into the folder, you can **open the folder and see the copied files**. GitHub Desktop offers a convenient button to open the folder, but of course, you can always use Windows Explorer or Finder to locate the files (Figure 20).

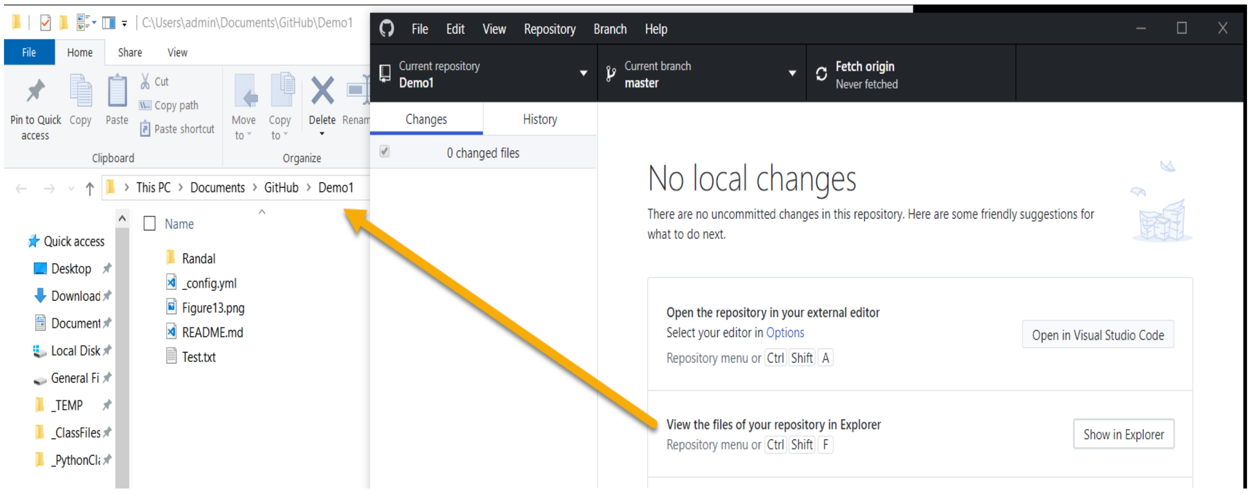
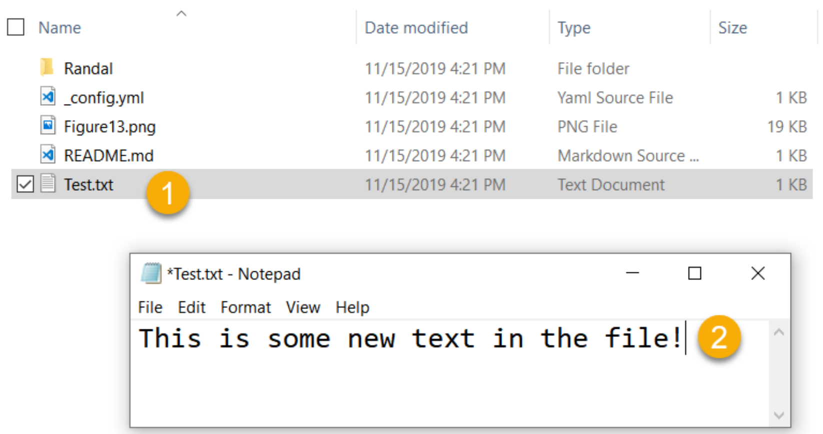


Figure 20. Viewing the local files

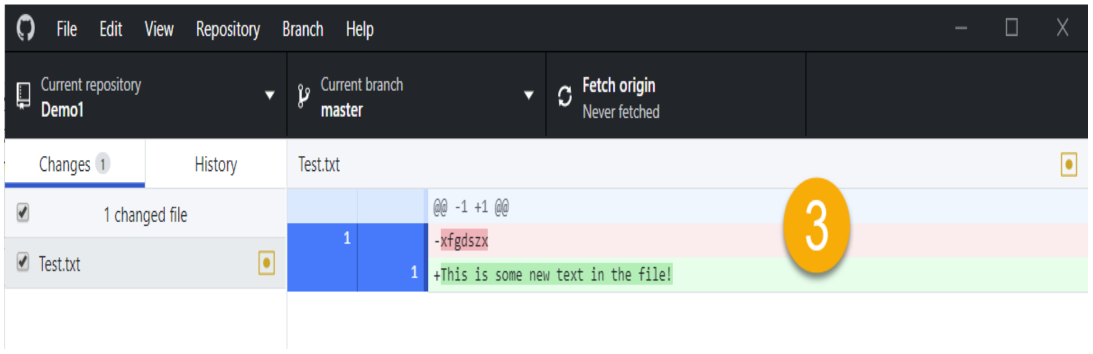
It may not look like it, but Git is now managing the folder! To see it in action, open a file, make a change to it, the go back to GitHub Desktop. The figure below shows some text I added to a text file. Figure 21 shows how GitHub Desktop displays the previous text in red and the new text in green (Figure 22.)



| Name | Date modified | Type | Size |
|---|---|---|---|
| Randal | 11/15/2019 4:21 PM | File folder | |
| _config.yml | 11/15/2019 4:21 PM | Yaml Source File | 1 KB |
| Figure13.png | 11/15/2019 4:21 PM | PNG File | 19 KB |
| README.md | 11/15/2019 4:21 PM | Markdown Source ... | 1 KB |
| Test.txt | 11/15/2019 4:21 PM | Text Document | 1 KB |

```
This is some new text in the file!
```

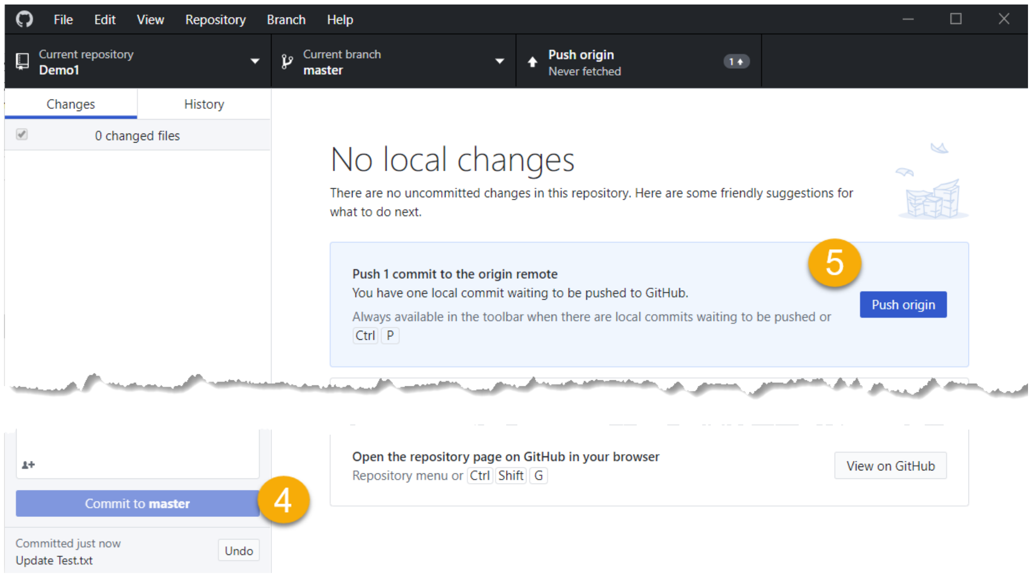
Figure 21. Changing text in a Git managed file



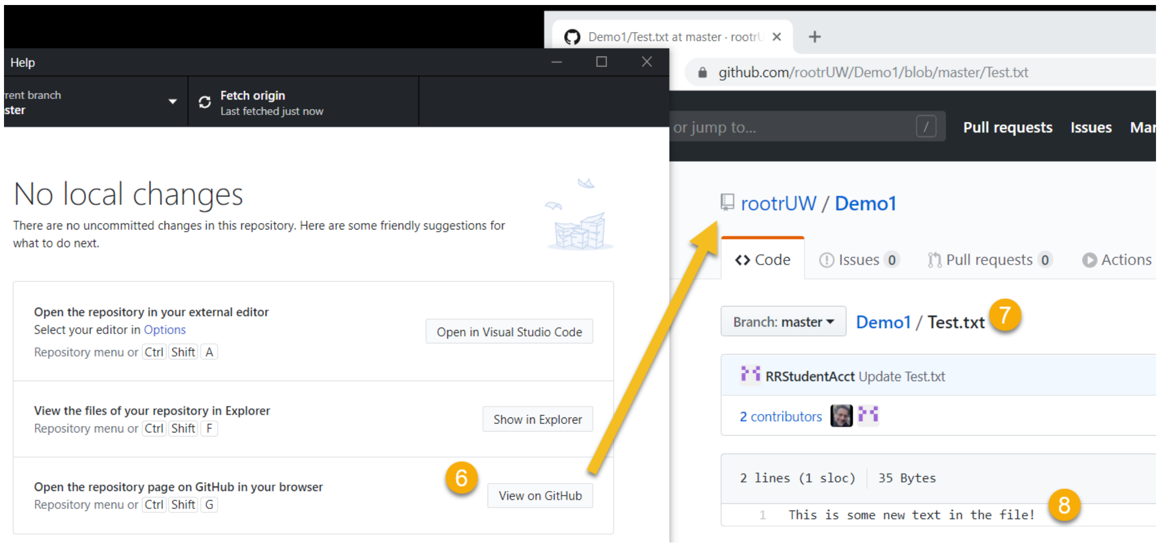
```
@@ -1 +1 @@
1    -xfgdszx
   1 +This is some new text in the file!
```

Figure 22. Reviewing the changes to a file in GitHub Desktop

If you want to upload your changes to the Main folder of your GitHub repository, click the "Commit to Main" button, then click the "Push origin" button when it appears (Figure 23.)


File
Edit
View
Repository
Branch
Help
Current repository
Demo1
Current branch
master
Push origin
Never fetched
Changes
History
0 changed files
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Push 1 commit to the origin remote
You have one local commit waiting to be pushed to GitHub.
Always available in the toolbar when there are local commits waiting to be pushed or Ctrl P
Push origin
5
Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub
Commit to master
4
Committed just now
Update Test.txt
Undo
Figure 23. Uploading the changes to GitHub

Once the upload is complete, you can navigate to GitHub and verify that both the local and website version of the file are the same (Figure 24.)


Help
Current branch
master
Fetch origin
Last fetched just now
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code
View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer
Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub
6
Demo1/Test.txt at master · rootr
github.com/rootrUW/Demo1/blob/master/Test.txt
or jump to...
Pull requests
Issues
rootrUW / Demo1
Code
Issues 0
Pull requests 0
Actions
Branch: master
Demo1 / Test.txt
7
RRStudentAcct Update Test.txt
2 contributors
2 lines (1 sloc) 35 Bytes
8

```
1  This is some new text in the file!
```

Figure 24. Viewing GitHub's website to verify the files are uploaded.

After that test is complete, you can start exploring the features of GitHub Desktop. You can learn more about the application on this GitHub website (https://help.github.com/en/desktop).

## Summary

In this module, we looked at how to use custom functions and try-except blocks to organize file management code and provided custom error handling. We also looked at ways you can improve your GitHub webpages to look more professional.

At this point, you should try and answer as many of the following questions as you can from memory, then review the subjects that you cannot. Imagine being asked these questions by a co-worker or in an interview to connect this learning with aspects of your life.

- What is the difference between a class and the objects made from a class?
- What are the components that make up the standard pattern of a class?
- What is the purpose of a class constructor?

- When do you use the keyword "self?"
- When do you use the keyword "@staticmethod?"
- How are fields and attributes and property functions related?
- What is the difference between a property and a method?
- Why do you include a docstring in a class?
- What is the difference between Git and GitHub?
- What is GitHub Desktop?

When you can answer all of these from memory, it is time to complete the module's assignment and move on to the next module.