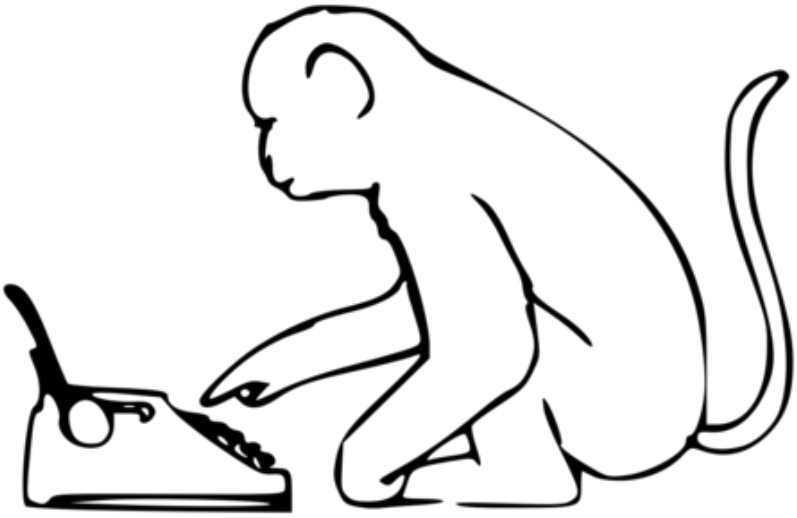
How to Write a Class in Python

In python, classes can help to wrap up data and functionality at the same time. Several classes have already been written for us in python 3, called builtins. Here are a few: int (integer class), str (string class), list (list class). This...

Part 1 of 2:

Introduction to Classes

1. 

   Open Python IDE. You can learn how to do this in Install Python.

2. **Use keyword `class`, followed by a space, the name of the class, and a colon.**

   ```
   class Duck:
   ```

3. **Indent and add basic variables for class.** To do this, press ? Enter or ? Return. Indent and write out a basic variable followed by an equal sign, and then your variable surrounded in quotes.

   ```
   class Duck: says = "Quack" gender = "Male" name = "Richie"
   ```

4. **Access the variables by creating instances of the class.**
   1. In python, the dot notation is used to access methods and/or variables defined in the class.
   2. An example is shown below.

```
class Duck: says = "Quack" gender = "Male" name = "Richie" myDuck =
Duck() # Create instance of Duck class what = myDuck.says
# Will access says variable of class Duck and
# assign it to the variable "what" print(what) # Will print "Quack"
```

5. **Add functions to the class (these are called methods of the class).**
    1. This is where the functionality of classes and their ability to store values can be seen.

```
class Duck: says = "Quack" gender = "Male" name = "Richie" def fly():
print('Duck flies')
```

6. **Call the method of the class; in this case, Duck.**
    1. Methods use the dot notation as well:
    2. Just like a normal function, use parenthesis to call the method of `myDuck`

```
class Duck: says = "Quack" gender = "Male" name = "Richie" def fly():
print('Duck flies') my_Duck = Duck() my_Duck.fly()
# Will print "Duck flies"
```

7. **Change attributes of class.**

```
class Duck: says = "Quack" gender = "Male" name = "Richie" def fly():
print('Duck flies') my_Duck = Duck() my_Duck.gender = "Female"
# Changes value of variable gender in my_Duck
# Now, printing my_Duck.gender will output "Female"
```

8. **Initialize the Class.** Classes run an initializing function every time the programmer creates an instance of that class.
    1. To create this function, add a few spaces between the first and second lines of the class and type **`def __init__(self):`** on the second line (make sure to indent).
    2. In the Duck example (`self` explained below):

```
class Duck: def __init__(self): self.says = 'Quack' self.gender =
"Male" self.name = "Richie" def fly(): print('Duck flies') my_Duck =
Duck() # You can still get the variables the same way, but now
# they are wrapped in a function - later they will be changed
# by other functions in class Duck.
```

The **`self`** word is the instance of the Duck class that is being created. This word can be whatever the programmer wishes as long as it is the first argument of the `__init__` function.

9. **Add Default Arguments to `__init__` function.** A class that does not take arguments of any kind is clunky. First, type this into the python console after the class definition:

```
class Duck: def __init__(self): self.says = 'Quack' self.gender =
"Male" self.name = "Richie" def fly(): print('Duck flies') my_Duck =
Duck() my_Duck.says = 'I don't want to quack' my_Duck.gender = "Female"
 my_Duck.name = 'Lizz' new_Duck = Duck() new_Duck.name = 'Dude'
new_Duck.says = "IDK"
```

There is a much better way to do the same process - in one line. This will require a little manipulation of the Duck class:

```
class Duck: def __init__(self, says='Quack', gender='Male', name=
'Richie'): self.says = says self.gender = gender self.name = name def
fly(): print('Duck flies')
```

Let's delve into this example, beginning with the arguments:

1. `says='Quack', gender='Male', name='Richie'` - these are *default arguments* - if the programmer inputs something else into the function, the argument will take that value instead. If the programmer doesn't input anything, the argument takes on the value assigned to it by the = operator.
2. Finally, the variables are added to the instance of the class that is created when the programmer calls the class method.

10. **Create Instances of Class with Default Variables.** For this example, we will re-create the two previous Ducks - my_Duck and new_Duck.

```
class Duck: def __init__(self, says='Quack', gender='Male', name=
'Richie'): self.says = says self.gender = gender self.name = name def
fly(): print('Duck flies') my_Duck = Duck('I don't want to quack',
'Female','Lizz') new_Duck = Duck('IDK', name = 'Dude')
# or new_Duck = Duck('IDK', 'Male', 'Dude') ''' Previous "chunky" code
my_Duck = Duck() my_Duck.says = 'I don't want to quack'
my_Duck.gender = "Female" my_Duck.name = 'Lizz' new_Duck = Duck()
new_Duck.name = 'Dude' new_Duck.says = "IDK"'''
```

Part 2 of 2:

Advanced Numerical Classes

1. **Begin the Class.** This was discussed in Part 1 of this article. For our example, we will write a fraction class:

```
def GCF(n, m):
# Using the Euclidean Algorithm to find the greatest common factor
while n: m, n = n, m % n return m def reduce_fraction(numerator,
denominator): g = GCF(numerator, denominator) numerator //= g
denominator //= g return numerator, denominator class Fraction: def
__init__(self, numerator, denominator = 1): self.fraction =
reduce_fraction(numerator, denominator) myFrac = Fraction(3, 4)
# Fraction of 3/4, will not be reduced print(myFrac)
```

Output:

2. **Overwrite the __str__ and __repr__ methods.** These two methods control how the instances of the class are displayed using the print function. A good programmer wants the fraction displayed when he/she types in `print(myFrac)`. Thus the following addition is made:

```
def GCF(n, m):
# Using the Euclidean Algorithm to find the greatest common factor
while n: m, n = n, m % n return m def reduce_fraction(numerator,
denominator): g = GCF(numerator, denominator) numerator //= g
denominator //= g return numerator, denominator class Fraction: def
__init__(self, numerator, denominator = 1): self.fraction =
reduce_fraction(numerator, denominator) def __str__(self): return str(
self.fraction[0]) + '/' + str(self.fraction[1]) __repr__ = __str__
# Assign one function to another.
# This is legal in python. We just renamed # __str__ with __repr__
myFrac = Fraction(6, 4) # Fraction of 6/4, will be reduced to 3/2 print
(myFrac)
```

Output:

```
3/2
```

3. **Add Functionality.** Please refer to the Official Python Docs for a complete list of operators that can be written as functions. For the Fraction class example, we will extend the class with an addition function. The two functions that need to be written to add classes together are the __add__ and __radd__ functions.

```
def GCF(n, m):
# Using the Euclidean Algorithm to find the greatest common factor
while n: m, n = n, m % n return m def reduce_fraction(numerator,
denominator): g = GCF(numerator, denominator) numerator //= g
denominator //= g return numerator, denominator def lcm(n, m): return n
 // GCF(n, m) # or m // GCF(n, m) def add_fractions(Frac1, Frac2):
denom1 = Frac1[1] denom2 = Frac2[1] Frac1 = Frac1[0] * denom2 Frac2 =
Frac2[0] * denom1 return reduce_fraction(Frac1+Frac2, denom1 * denom2)
class Fraction: def __init__(self, numerator, denominator = 1): self.
fraction = reduce_fraction(numerator, denominator) def __str__(self):
return str(self.fraction[0]) + '/' + str(self.fraction[1]) __repr__ =
__str__ # Assign one function to another.
# This is legal in python. We just renamed # __str__ with __repr__ def
__add__(self, other_object): if isinstance(other_object, int):
# if other_object is an integer return self + Fraction(other_object)
# Make it of the Fraction class
# (integers are just fractions with 1 as the denominator, after all!)
if isinstance(other_object, Fraction): return add_fractions(self.
fraction, other_object.fraction) else: raise TypeError(
"Not of class 'int' or class 'Fraction'") myFrac = Fraction(6, 4)
# Fraction of 6/4, will be reduced to 3/2 other_Frac = Fraction(2,3)
print(myFrac + other_Frac, 'n') print(myFrac + 2)
```

Output:

```
13/6 7/2
```

4. **Continue Looking Around.** This article has just scratched the surface on what classes can do. Another great resource for any questions is Stack OverFlow. For a challenge, go to Think Functional and write the classes.