CoGrammar

Lecture 7: Classes

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(FBV: Mutual Respect.)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Open Classes. You can submit these questions here: **Open Class Questions**

Lecture Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Lecture Objectives

- Grasp the basic concepts of Object Oriented Programming
- Learn how to implement classes and use them

CoGrammar

Object Oriented Programs

★ OOP is a programming language feature that allows you to group variables and functions together into new data types, called classes, from which you can create objects.

★ By organizing your code into classes, you can break down a monolithic program into smaller parts that are easier to understand and debug.

CoGrammar

Object Oriented Programs

★ For small programs, OOP doesn’t add organization so much as it adds bureaucracy.

★ Although some languages, such as Java, require you to organize all your code into classes, Python OOP features are optional.

Real-World Analogy

★ You’ve most likely had to fill out paper or electronic forms numerous times in your life: for doctor’s visits, for online purchases, or to RSVP to a wedding.

★ Forms exist as a uniform way for another person or organization to collect the information they need about you.

CoGrammar

Real-World Analogy

★ Different forms ask for different kinds of information.

★ You would report a sensitive medical condition on a doctor’s form, and you would report any guests you’re bringing on a wedding RSVP, but not the other way around.

Real-World Analogy

★ In Python, class, type, and data type have the same meaning.

★ Like a paper or electronic form, a class is a blueprint for Python objects (also called instances), which contain the data that represents a noun.

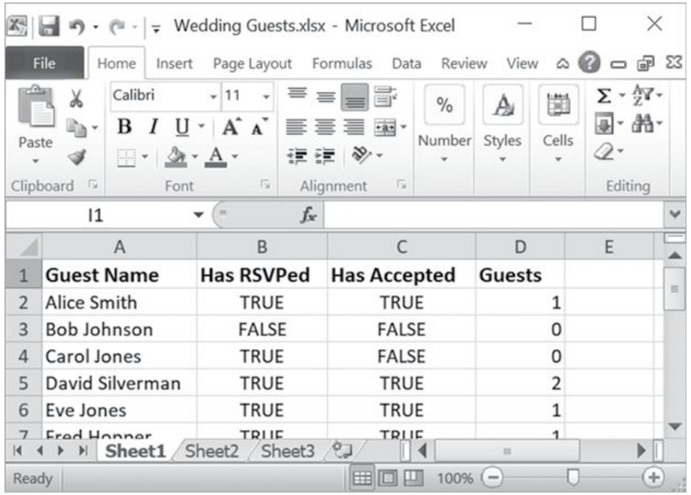
Real-World Analogy

★ This noun could be a doctor’s patient, an ecommerce purchase, or a wedding guest.

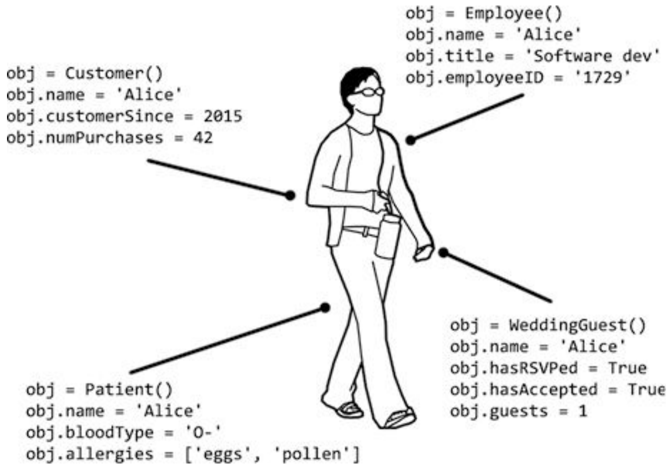
★ Classes are like a blank form template, and the objects created from that class are like filled-out forms that contain actual data about the kind of thing the form represents.

Real-World Analogy

- ★ You can also think of classes and objects as spreadsheets
- ★ The column headers would make up the class, and the individual rows would each make up an object.



	A	B	C	D	E
1	Guest Name	Has RSVPed	Has Accepted	Guests	
2	Alice Smith	TRUE	TRUE	1	
3	Bob Johnson	FALSE	FALSE	0	
4	Carol Jones	TRUE	FALSE	0	
5	David Silverman	TRUE	TRUE	2	
6	Eve Jones	TRUE	TRUE	1	
7	Fred Hopper	TRUE	TRUE	1	



```
obj = Employee()
obj.name = 'Alice'
obj.title = 'Software dev'
obj.employeeID = '1729'
```

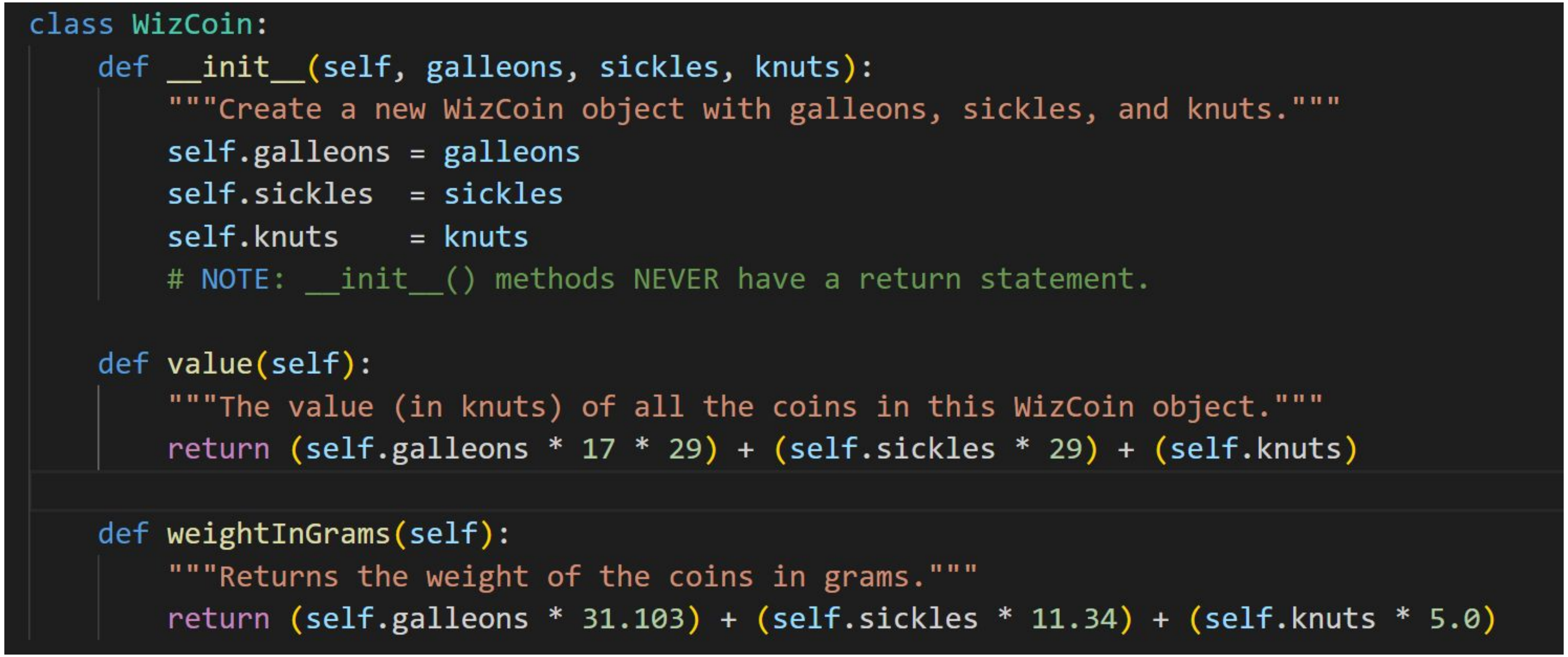
```
obj = Customer()
obj.name = 'Alice'
obj.customerSince = 2015
obj.numPurchases = 42
```

```
obj = WeddingGuest()
obj.name = 'Alice'
obj.hasRSVPed = True
obj.hasAccepted = True
obj.guests = 1
```

```
obj = Patient()
obj.name = 'Alice'
obj.bloodType = 'O-'
obj.allergies = ['eggs', 'pollen']
```

CoGrammar

Creating a class



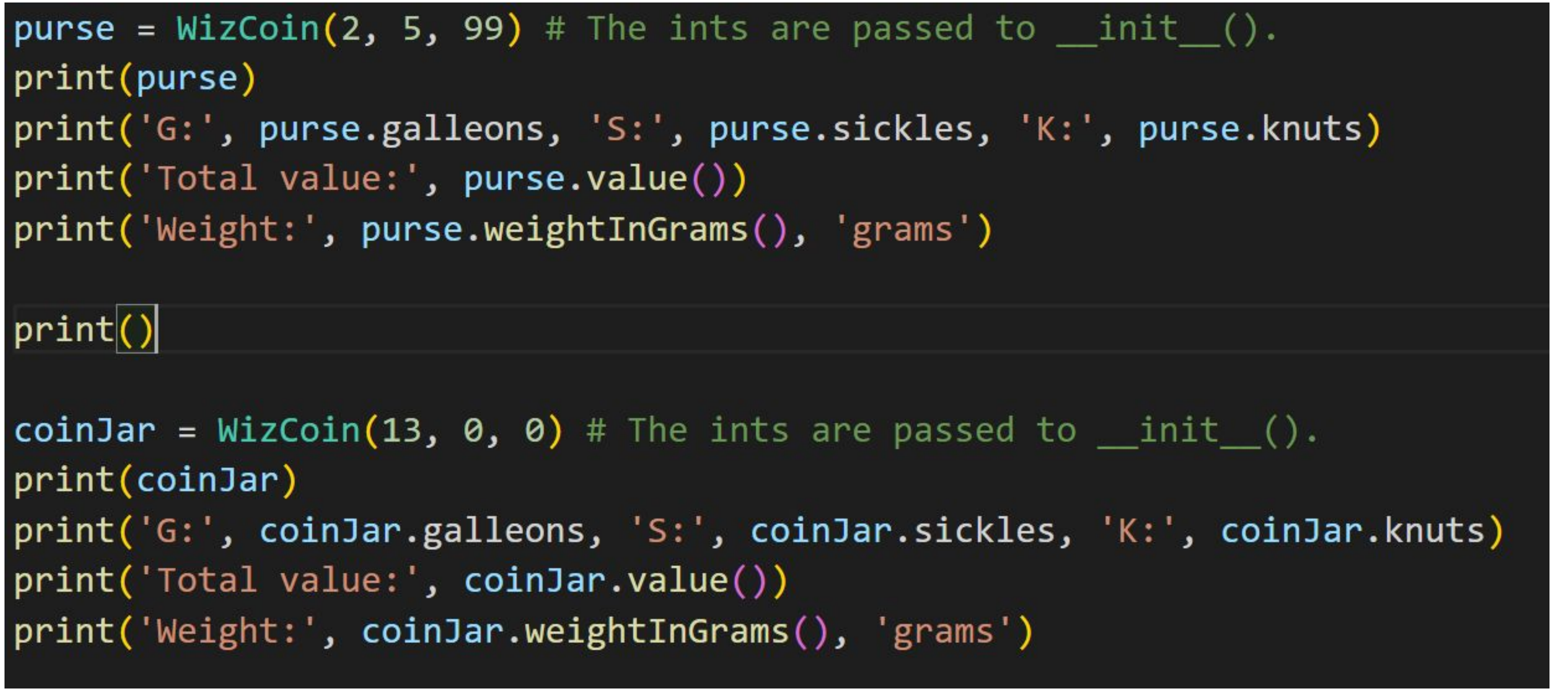
```python
class WizCoin:
    def __init__(self, galleons, sickles, knuts):
        """Create a new WizCoin object with galleons, sickles, and knuts."""
        self.galleons = galleons
        self.sickles  = sickles
        self.knuts    = knuts
        # NOTE: __init__() methods NEVER have a return statement.

    def value(self):
        """The value (in knuts) of all the coins in this WizCoin object."""
        return (self.galleons * 17 * 29) + (self.sickles * 29) + (self.knuts)

    def weightInGrams(self):
        """Returns the weight of the coins in grams."""
        return (self.galleons * 31.103) + (self.sickles * 11.34) + (self.knuts * 5.0)
```

CoGrammar

Using a class



```python
purse = WizCoin(2, 5, 99) # The ints are passed to __init__().
print(purse)
print('G:', purse.galleons, 'S:', purse.sickles, 'K:', purse.knuts)
print('Total value:', purse.value())
print('Weight:', purse.weightInGrams(), 'grams')

print()

coinJar = WizCoin(13, 0, 0) # The ints are passed to __init__().
print(coinJar)
print('G:', coinJar.galleons, 'S:', coinJar.sickles, 'K:', coinJar.knuts)
print('Total value:', coinJar.value())
print('Weight:', coinJar.weightInGrams(), 'grams')
```

CoGrammar

Methods, __init__(), and self

★ Methods are functions associated with objects of a particular class. Recall that lower() is a string method, meaning that it’s called on string objects.

★ Also, notice that methods come after the object.

Methods, __init__(), and self

★ We create objects by calling the class name as a function. This function is referred to as a constructor function (or constructor, or abbreviated as ctor, pronounced “see-tore”) because it constructs a new object.

★ We also say the constructor instantiates a new instance of the class.

CoGrammar

Methods, __init__(), and self

★ Calling the constructor causes Python to create the new object and then run the __init__() method.

★ Classes aren't required to have an __init__() method, but they almost always do.

★ The __init__() method is where you commonly set the initial values of attributes.

★ When a method is called on an object, the object is automatically passed in for the self parameter.

Methods, __init__(), and self

★ The rest of the arguments are assigned to parameters normally.

★ You don’t have to name a method’s first parameter self; you can name it anything. But using self is conventional.

OOP is a programming language feature that...

A. Stores variables and functions into classes
B. Stores items in a list
C. Stores key value pairs
D. Stores numbers in a variable

What are methods?

A. Name of a band
B. Functions associated with objects of a class
C. Variables associated with objects of a class
D. Colors associated with objects of a class

How to create objects in Python?

A. Calling the class name as a function.
B. Using the new keyword
C. Using the self keyword
D. Object objectname

What do we call a function that is used to create an object?

A. Type function
B. Power function
C. String function
D. Constructor function

How to create a object variable?

A. self.health
B. int.health
C. str.health
D. bool.health

Let’s Breathe

Let’s take a small break before moving on to the next topic.

CoGrammar

CoGrammar

Attributes

★ **Attributes are variables associated with an object.**

★ **For example, consider the** *birthday.year* **expression. The** *year* **attribute is a name following a dot.**

★ **Every object has its own set of attributes.**

CoGrammar

Attributes

★ You can think of an object's attributes as similar to a dictionary's keys.

★ You can read and modify their associated values and assign an object new attributes.

★ Technically, methods are considered attributes of a class, as well.

Private Attributes and Private Methods

- ★ In languages such as C++ or Java, attributes can be marked as having private access, which means the compiler or interpreter only lets code inside the class's methods access or modify the attributes of objects of that class.
- ★ But in Python, this enforcement doesn't exist. All attributes and methods are effectively public access: code outside of the class can access and modify any attribute in any object of that class.

Private Attributes and Private Methods

★ **Private access is useful.**

★ **For example, objects of a BankAccount class could have a balance attribute that only methods of the BankAccount class should have access to.**

Private Attributes and Private Methods

★ For those reasons, Python’s convention is to start private attribute or method names with a single underscore.

★ Technically, there is nothing to stop code outside the class from accessing private attributes and methods, but it’s a best practice to let only the class’s methods access them.

CoGrammar

type() Function

★ Passing an object to the built-in type() function tells us the object's data type through its return value.

★ The objects returned from the type() function are type objects, also called class objects.

★ Recall that the terms type, data type, and class all have the same meaning in Python.

CoGrammar

Inheritance

★ Inheritance is a code reuse technique that you can apply to classes.

★ It's the act of putting classes into parent-child relationships in which the child class inherits a copy of the parent class's methods, freeing you from duplicating a method in multiple classes.

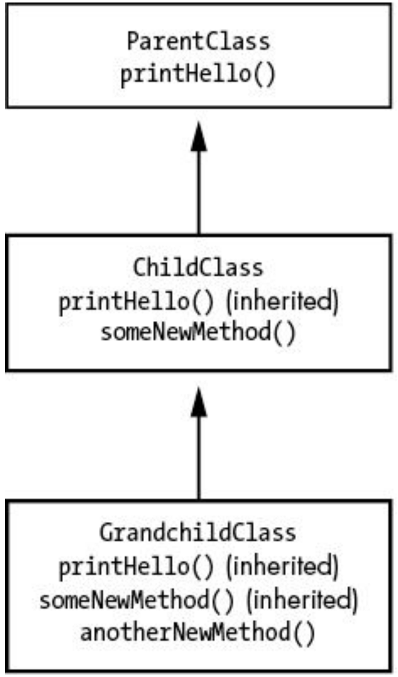
★ To create a new child class, you put the name of the existing parent class in between parentheses in the class statement.

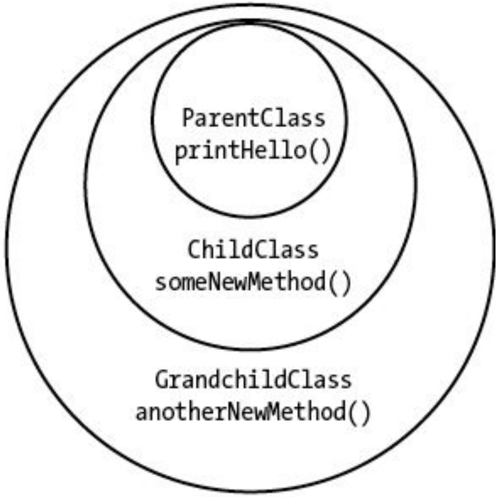
Inheritance

- ★ It’s common to say that parent-child classes represent “is a” relationships.
- ★ A ChildClass object is a ParentClass object because it has all the same methods that a ParentClass object has, including some additional methods it defines.
- ★ We also sometimes call a child class a subclass or derived class and call a parent class the super class or base class.

CoGrammar

Inheritance





What is inheritance?

A. Putting classes into parent-child relationships
B. Turning classes into friends
C. Destroying objects
D. Adding attributes to a class

Passing an object to the built-in type() function tells us...

A. The data type of the object
B. The number of attributes in the object
C. The number of dictionaries in object
D. Size of the memory consumed by the object

All attributes and methods are effectively public access.

A. True
B. False

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

CoGrammar

Thank you for joining!