# CSCA08 Fall 2017

## Week 8 - Intro to OOP

Brian Harrington & Marzieh Ahmadzadeh

University of Toronto Scarborough

October 30 - November 3, 2017

UNIVERSITY OF TORONTO SCARBOROUGH

# Admin

- TT2
  - Details on course website
  - Covers everything up to and including dictionaries (week 7 lecture, week 8 tutorials + inverted)
  - Beware the code mangler

# Defining our own Types

- We've seen a bunch of types so far (`int`, `float`, `list`, `dict`...).
- Now it's time to define our own

# Object Orientation: A New Paradigm

- Until now: Functions were the focus
  - `my_function(data)`: One global function that gets data passed to it
- Object Oriented Approach:
  - `my_obect.method(data)`: The object has its own methods and data

# Some terminology

- Class: The type of an object
- Object: An instance of a class
- Method: Like a function, but belongs to a class
- We've already seen this:
  - `my_string = str(12.57)`
  - `my_string.ljust(10)`
  - `my_string` is an *object*, of the *str* class, and we called string's *ljust* method on it

# Creating Objects

- Create a class: `class ClassName():`
  - (We'll see what the brackets are for later)
  - CamelCase (not pot_hole_case)
- define a method `def method_name(self)`
  - Looks very similar to defining a function
  - Indented *inside* the class
  - We'll see what the `self` does in a minute
- can now create a new object of type ClassName
  - `my_object = ClassName()`
- our new object can now access the methods we defined
  - `my_object.method_name()`

# Self

- Every method (including the built in ones) gets implicitly passed a copy of the object upon which it was called
- We don't include it in the method call, but we do in the method definition
- This allows a method to access the object on which it was called
- We normally call this copy of our object `self`
- Behind the scenes: `my_obj.method()` is really just an alias of `Class.method(my_obj)`

# Break

WARNING: Scary Halloween Image Ahead

# Break

# What's With all the Underscores?

- Already saw, we can do: `my_object.variable`
- This is bad. Why?
  - External code relying on internals of class
  - We should be able to change internals without worrying about breaking external code
  - Like the difference between internal/external documentation
  - This will be a major topic in A48
- using underscores = people unlikely to guess variable names
  - Really: Just a way of saying "Hands off my variables"
  - Doesn't actually stop external code, if they know the variable name, they can still access/change it.
  - Security through obscurity/convention

# Built-in Methods

- __init__
  - Initialize: this is known as a constructor method
  - Defines the code that runs when we first create a new object of this type
  - Usually used to set up the default parameters
- __str__
  - Return what you want to output when an object of this class is cast to a string (or printed)