CSC148-Section:L0301 Week#1-Monday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap and Jacqueline Smith slides winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Announcements
- Objects
- Functions
  - CSC108 Recipe for Designing Functions
- Classes
  - CSC148 Recipe for Designing Classes

Announcements

- The **information sheet draft** for one week (and then it becomes permanent)
  - and if there are any big issues, address them ***now***
    - E.g. check ***now*** whether you have any test conflicts with other courses
- You should have got email "**CSC148 tutorial/lab for January 11/12**"
  - http://www.teach.cs.toronto.edu/~csc148h/winter/Labs/lab1/handout.pdf
  - If NOT check your portal and try to fix this problem
- You **must be signed up** for a tutorial time slot in order to attend a lab and get your lab mark.

Objects

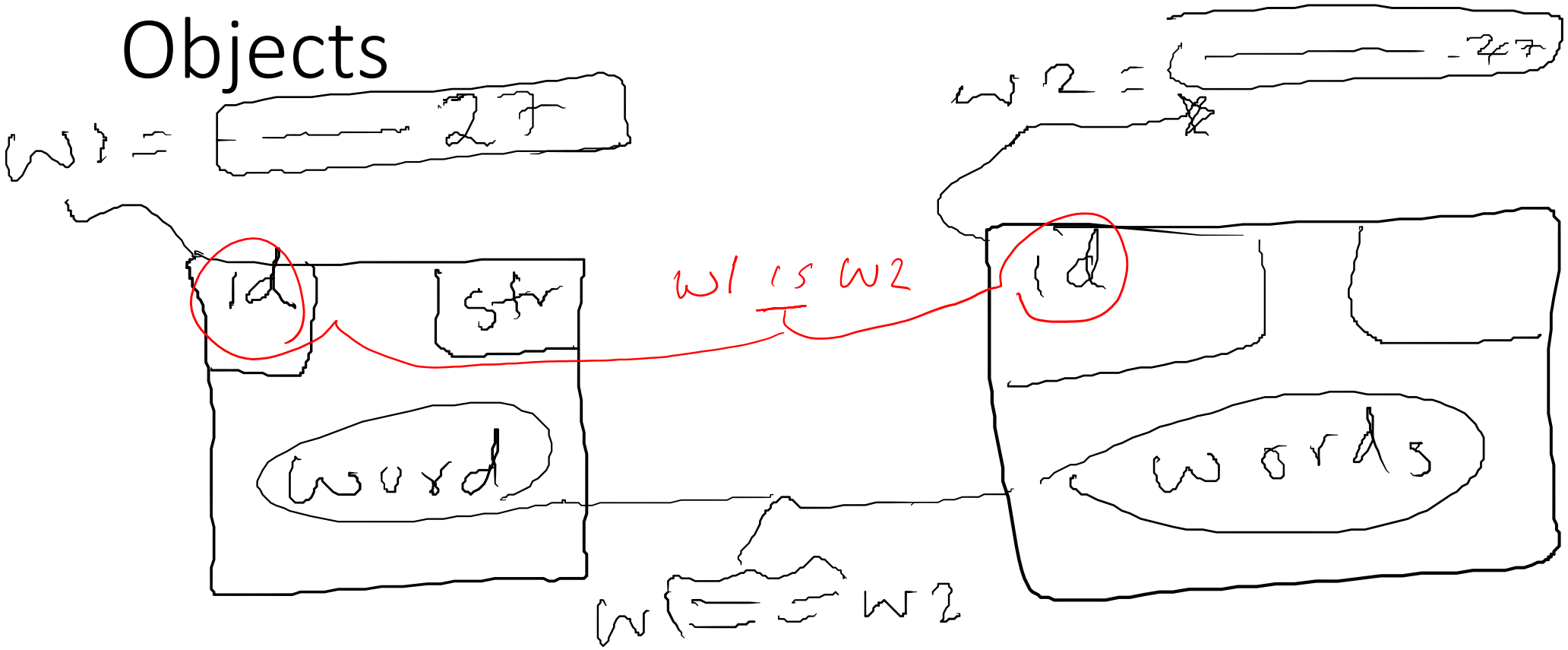
some built-in objects to fool around with:

```
>>> w1 ='words'
>>> w2='swords'[1:]
>>> w1 is w2
False
>>> id(w1)
1377274355472
>>> id(w2)
1377303354424
>>> w1 is w2
False
>>> w1 == w2
True
>>> type(w1)
<class 'str'>
```

```
>>> w1 *w2
```

What is the output?

Objects



Functions :
CSC108 Recipe for Designing Functions

1. Examples
2. Header
3. Description
4. Body
5. Test Your Function

Computer Science
UNIVERSITY OF TORONTO

1. Examples

- Pick a name for the function (often a verb or verb phrase)
  - a good name is a short answer to the question “What does your function do?”
- Write one or two examples of calls to your function and the expected returned values
  - Do not include examples for functions that involve randomness or I/O
- Include an example of
  - a standard case
  - as opposed to a tricky or corner case.
- Put the examples inside a triple-quoted string that you’ve indented since it will be the beginning of the docstring.

```
"""
>>> is_even(2)
True
>>> is_even(17)
False
"""
```

2. Header

- Write the function header above the docstring and outdent it
- Choose a meaningful name for each parameter (often nouns).
- Include the type contract (the types of the parameters and x value).

```
def is_even(value: int) -> bool:
    """
    >>> is_even(2)
    True
    >>> is_even(17)
    False
    """
```

Computer Science
UNIVERSITY OF TORONTO

3. Description

- Before the examples, add a description of what the function does and mention each parameter by name.
- Describe the return value.

```
def is_even(value: int) -> bool:
    """Return True if and only if value is evenly divisible by 2.
    >>> is_even(2)
    True
    >>> is_even(17)
    False
    """
```

4. Body

- Write the body of the function by remembering to indent it to match the docstring.
- To help yourself write the body, review your example cases from step 1 and consider how you determined the return values
- You may find it helpful to write a few more example calls.

```
def is_even(value: int) -> bool:
    """Return True if and only if value is evenly divisible by 2.
    >>> is_even(2)
    True
    >>> is_even(17)
    False
    """

    return value % 2 == 0
```

5. Test Your Function

- Test your function on all your example cases including any additional cases you created in step 4.
- try it on extra tricky or corner cases.

```
def is_even(value: int) -> bool:
        """Return True if and only if value is evenly divisible by 2.
        >>> is_even(2)
        True
        >>> is_even(17)
        False
        """
        return value % 2 == 0


if __name__ == '__main__':
    from doctest import testmod
    testmod()
```

Example

- Write a function that accepts the number of pizzas that you are ordering and the number of slices per pizza, and returns the total number of slices in the order.

Example

- Write a function that accepts the number of pizzas that you are ordering and the number of slices per pizza, and returns the total number of slices in the order.

```
def total_slices(num_pizzas: int, slices_per_pizza: int) -> int:
    """Return the total number of slices in num_pizzas pizzas that each have
    slices_per_pizza slices.
    >>> total_slices(1, 8)
    8
    >>> total_slices(3, 12)
    36
    """
    return num_pizzas * slices_per_pizza
```

Classes and Objects

- "Classes provide a means of **bundling data and functionality** together. Creating a new class **creates a new *type* of object**, allowing new *instances* of that type to be made."[1]
- "Each class instance can have **attributes** attached to it for maintaining its state. Class instances can also have **methods** (defined by its class) for modifying its state." [1]

- Let us see the **CSC148 Recipe for Designing Classes**

Computer Science
UNIVERSITY OF TORONTO

[1] https://docs.python.org/3/tutorial/classes.html

CSC148 Recipe for Designing Classes

- Part 1: Define the **API** for the class
- Part 2: **Implement** the class

- Exercise the recipe by building Point class
  - http://www.teach.cs.toronto.edu/~csc148h/winter/lecturedata/Danny/W1/point_exercise.pdf

Part 1: Define the API for the class

1. **Class name and description.**
   - Pick a noun to be the name of the class, write a one-line summary of what that class represents, and (**optionally**) write a longer, more detailed description.

```
class Student:
    """ Represents an enrolled student.
    """
```

2. **Example.**
   - Write some simple examples of client code that uses your class.

```
s1 = Student('Alice', 2018)
s2 = Student('Mike', 2017)
s1.calculate_GPA()
s1.set_address('50 St George')
s1 == s2
print(s1)
s1 is s2
```

Part 1: Define the API for the class

3. Public methods.

- Decide what services your class should provide. For each, define the API for a method that will provide the action.

  __init__
  __eq__
  __str__
  calculate_GPA
  set_address

- Use the first four steps of the Function Design Recipe to do so:
  (1) Example
  (2) Type Contract
  (3) Header
  (4) Description

Computer Science
UNIVERSITY OF TORONTO

Part 1: Define the API for the class

4. Public attributes.

Decide what data you would like client code to be able to access without calling a method.

name: str

id: int

Add a section to your class docstring after the longer description, specifying the **type**, **name**, and **description** of each of these attributes.

```
=== Attributes ===
@type name: str
    student's name
@type id: int
    student's id
"""
```

Part 2: Implement the class

1. Internal attributes

- Define the **type**, **name**, and **description** of each of the internal attributes. Put this in a class comment (using the hash symbol) below the class docstring.

2.Public methods.

- Use the last two steps of the function design recipe to implement all of the methods:

  (5) Code the Body

  (6) Test your Method

Exercise: building Point class

Somewhere in the real world there is a description of points in two-dimensional space:

> In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle.

Exercise: building Point class

Find the most important noun (good candidate for a class. . .), its most important attributes, and operations that sort of noun should support.

On the back of this sheet are steps taken from the recipe for designing Python classes. You'll be using this in more depth during lab, but for now we'll use it as a guide to creating the Point class. Here are exercises to carry out on paper, perhaps you'll need some scrap paper as well as this hand-out. You are, of course, encouraged, to continue the exercise on a computer in Python.

1. Carry out steps 1 and 2 of Part 1 for the description of points above.

2. Carry out steps 3 and 4 of Part 1 for the description of points above.

3. Carry out step 2 of Part 2.

Exercise: building Point class

- Download and Complete the missing parts in point_api.py:
  - http://www.cdf.utoronto.ca/~csc148h/winter/lecturedata/Danny/W1/point_api.py