Classes 3

Marquette University

Address Class

- How to generate addresses
  - Each country has its own way of generating addresses
  - An address consists of
    - an optional modifier (apartment, floor, neighborhood)
    - a street
    - a street number
    - a city
    - a state (in most of the Americas)
    - a country

Address Class

- To deal with optional arguments:
  - Use a default argument of none

```
def __init__(self, country,  city, street, number,
             postal,    state, apartment = None):
```

Aside: How to deal with long lines in Python

- Python statements ideally fit in a single line
- In fact, if you want to write poorly readable code, you can put more than one statement in a line and separate with a semi-colon (;)
- Python still allows to use a single forward slash as a continuation marker
- But this is not very readable
- Put expressions into parentheses (unless they already come with parentheses)
- Python interpreter will interpret correctly

The purpose of str and repr

- The dunder methods __str__ and __repr__ seem to do the same thing,
    - But:
        - __str__ is called by print with priority over __repr__
            - This is how you want your output be displayed
        - __repr__ should represent the internal structure of your class instances

Addresses

- We can use __repr__ to just give us the internal makeup of an Address instance

```
def __repr__(self):
        return "apartment: {0}\nstreet: {1}\nnumber: {2}\ncity: {3}\npostal: {4}\nstate: {5}, \ncountry: {6}".format(
            self.apartment, self.street, self.number, self.city, self.postal, self.state, self.country)
```

Addresses

- But for __str__, we will let the country code determine what to do.
- The code is ugly, but that is the price for internationalization
- And we have not even discussed how to be able to use non-English keyboard letters in Python

Self Test

- Open up the file address.py
  - Edit the __str__ dunder method to allow for US addresses

Addresses

- When we use str(my_address) on an Address object, we get the result of __str__
- When we use repr(my_address), we get the result of __repr__

Instances can be fields of classes

- When we model processes (such as business processes), we will build up our entities from simpler entities
  - We can have a has-a relationship
    - For example, each person has an address
      - (With many sad exceptions: some have none, some have more than one)

Modular programming

- Remember modules:
  - They are just py-files
  - They are imported using import statements
  - The form of the import statements determines how the names are being resolved
  - `import address`
    - imports the module, names are prefixed with "`address.`"
  - `from address import *`
    - Not recommended, just use names without prefix
  - `from address import Address`
    - Just as before, but only imports the class Address

Client Example

- Clients have a name and an address

```
import address

class Client:
    def __init__(self, name, address):
        self.name = name
        self.address = address
    def __str__(self):
        return "{}\n{}".format(self.name, str(self.address))
    def __repr__(self):
        return "Name: {}\n {}".format(self.name, repr(self.address))

if __name__=="__main__":
    address4 = address.Address("Canada", "Ottawa", "Wellington Street",
                       80, "ON K1A 0A2", "Ontario",
                       "Office of the Prime Minister")
    trudy = Client("The Honorable Justin Trudeau", address4)
    print(trudy)
```