Lecture 14

Queries

1 Printing Objects

Some of you may have noticed that when you print an object (an instance of a class) like a point, Python does something weird/not useful. Python is telling us the location of the Point in computer memory. This could be useful sometimes but often it isn't very. Let's change it!

```
class Pet:
    def __init__(self, name, animal, age):
        self.name = name
        self.animal = animal
        self.age = age

    def __eq__(self, other):
        return (self.name == other.name and
                self.animal == other.animal and
                self.age == other.age)

    # Ask for a printing format.  Then make the __str__ method print
    # that format.  Print now prints a pretty format!
    def __str__(self):
        return ("Name: {:8s} Animal: {:9s} Age: {:2d}"
                .format(self.name, self.animal, self.age))

    # And while that's great and all, it's hard for Python to parse...
    # By which I mean, if someone gave you pet data like that, it
    # would be a pain.  It's such a poor representation of a Pet.
    # Let's make a better representation!
    def __repr__(self):
        return ("Pet('{:s}', '{:s}', {:d})"
                .format(self.name, self.animal, self.age))


if __name__ == '__main__':
    pets = []
    pets.append(Pet("Oliver", "Iguana", 12))
    pets.append(Pet("Charlie", "Mongoose", 7))
    pets.append(Pet("Spot", "Dog", 2))
    pets.append(Pet("Casper", "Cat", 5))

    for pet in pets:
        print(pet)
```

Play around with **str**/**repr** method. Show that **str** is called by print and **repr** is called on command line.

2 Dictionaries

Uncomment one section at a time

```
# ------------ Example 1 - Simple Dictionary ----------------------
dict1 = {
    "name": "Alex",
    "age": 13,
    "gpa": 3.9
}

print(dict1["name"])
print(dict1["age"])
print(dict1["gpa"])

gpa = dict1["gpa"]
print("GPA is", gpa)
print("\n\n")

# ------------ Example 2 - Dictionary with an Array ----------------
dict2 = {
    "name": "Alex",
    "age": 13,
    "gpa": 3.9,
    "classes": [
        "English",
        "Science",
        "PE",
        "Shop II",
        "Adv. Math",
    ]
}

classes = dict2["classes"]
for c in classes:
    print(c)

print("\n\n")

# ------- Example 3 - Dictionary with an Array of Dictionaries ------
dict3 = {
    "name": "Alex",
    "age": 13,
    "gpa": 3.9,
    "classes": [
        {
            "name": "English",
            "grade": 91.4,
            "letter_grade": "A-"
```

```
        },
        {
            "name": "Adv. Math",
            "grade": 88.2,
            "letter_grade": "B+"
        },
        {
            "name": "Shop II",
            "grade": 100.0,
            "letter_grade": "A"
        }
    ]
}

classes = dict3["classes"]
for c in classes:
    print(c["name"] + ": " + c["letter_grade"])

print("\nAlex's percentage in Shop II is:",
      dict3["classes"][2]["grade"])
```

3 Exceptions

What happens when I do the following?

```
value = int(input("Enter an integer: "))

print("You entered: {:d}.".format(value))
```

To ask a slightly different question, what could go wrong? What if I type in "five"? What happens then?

It crashes! More specifically, it raises an exception! More specifically, it raises a `ValueError` (this will be important in a moment). But what if I don't want the program to crash just because of a slight error? I can tell python to 'try' to do something and do something else of an exception is raised.

```
try:
    value = int(input("Enter an integer: "))

    # We only get here if there was not an error.
    print("You entered: {:d}.".format(value))
except ValueError:
    print("ERROR: That wasn't an integer.\n")
```

Let's modify the code so that I keep asking for a value until it's an integer.

```
is_valid_input = False

while not is_valid_input:
    try:
        value = int(input("Enter an integer: "))
        is_valid_input = True
```

```
    except ValueError:
        print("ERROR: Please enter an integer.\n")

print("You entered: {:d}.".format(value))
```

Here I'm saying to 'try' to do a possibly unsafe operation, and if a `ValueError` occurs, instead do something else.