Reminder: Lab_fundamentals due tonight!

No extensions for labs*

Use Piazza or office hours to get assistance.

Be sure your code is on Prairielearn and passes the autograder
Learning Objectives

Formalize and explore function overloading (in Python)

Introduce object-oriented programming

Discuss and practice defining interfaces for computational problems

Introduce class definitions in python
Python Functions

A function in Python is defined by its name and its input parameters

def getTotalTime(checkin, checkout):

def getSmallestEven(x, y, z):

def electricBill(watts):

print(getTotalTime("09:00:00","17:31:53"))

print(getSmallestEven(2, 1, 3))

print(electricBill(40))
Function Overloading

Two functions are **overloaded** when they have the same name but different parameters.

```python
# Example of function overloading

def combine(x, y):
    return [x, y]

print(combine(5, 1))

def combine(list1, list2):
    return list1+list2

print(combine([1, 2], [3, 4]))

def combine(x, list1, list2):
    return [x]+list1+list2

print(combine(0, [1, 2], [4, 5]))
```
Function Overloading

To properly define an overloaded function, give default arguments.

```python
def combine(x, y=None, list1 = None, list2 = None):
    out = [x]
    if y:
        out+=[y]
    if list1:
        out+=list1
    if list2:
        out+=list2
    return out

print(combine(5, 1))
print(combine(0, [1, 2], [4, 5]))
print(combine(0, list1=[1, 2], list2=[4, 5]))
```
Function Overloading

For true freedom of input, use keyword *args and **kwargs

```python
def combine(*args, **kwargs):
    out = []
    for a in args:
        out.append(a)
    for k, v in kwargs.items():
        print("{} = {}".format(k, v))
        out+=v
    return out

print(combine(0, 1, 2, 3, 4, \
              list1=[9, 2,3,1], list2=[8,7,2,1], \
              list3 = [10])))
```
Function Overloading

You might be wondering why this is important…

```python
import pandas

pd.read_table('myFile.csv')

pd.read_table('myFile.csv', delimiter=',')

pd.read_table('myFile.csv', delimiter=',', usecols=['Netid', 'Grade'])
```
Object-Oriented Programming

An **object** is a conceptual grouping of variables and functions that make use of those variables. A function associated with an object is a **method**.

Variables:

Methods:
Object-Oriented Programming

c1.area()
c2.xpos == c3.xpos
c2.ypos == c3.ypos
getTotalArea(c4, c5, ...
Object-Oriented Programming

An **object** is a conceptual grouping of variables and methods that make use of those variables. **You’ve been using these the entire time**

**Everything in Python is an object**

```
1   x = "myString"
2   print(x.capitalize())
3   print(x.find("String"))
4   print(x.upper())
5   print(x[3]) # __getitem__()
6   print(x) # __str__()
```

<table>
<thead>
<tr>
<th>Variables:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>String</td>
</tr>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>myString</td>
</tr>
<tr>
<td><strong>Ref Count</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**Methods:**
Object-Oriented Programming

Even things that don’t have obvious function calls are (secretly) defined as a method of some object.

```
1 a="3"
2 b=3
3 c=3.0
4 d=True
5 print(a + b)
6 print(b + c)
7 print(c > d)
```

```python
1 # For objects of type 'string'
2 def __add__(self, o):
3     ...
4
5 # For objects of type 'int'
6 def __add__(self, o):
7     ...
8
9 # For objects of type 'float'
10 def __add__(self, o):
11     ...
12
13 def __gt__(self, o):
14     ...
15
16
even things that don't have obvious function calls are (secretly) defined as a method of some object.
Object-Oriented Programming

The collection of publicly accessible methods and variables that make up an object is its **interface**. This includes none of the implementation details.

```python
str.join(iterable)
Return a string which is the concatenation of the strings in iterable. A TypeError will be raised if there are any non-string values in iterable, including bytes objects. The separator between elements is the string providing this method.

str.ljust(width[, fillchar])
Return the string left justified in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

str.lower()
Return a copy of the string with all the cased characters [4] converted to lowercase.

The lowercasing algorithm used is described in section 3.13 of the Unicode Standard.

str.lstrip([chars])
```

[4] Case in the context of strings refers to their capitalization (upper or lower case).

https://docs.python.org/3/library/stdtypes.html#string-methods
Object-Oriented Programming

We will discuss and use data structures in the context of their interface.

Ex: The string [data type] will have a few properties in any language.

```cpp
std::string x = "Hello World";
for(int i = x.length() - 1; i >= 0; --i){
    std::cout << x[i] << std::endl;
}
```
In-Class Exercise

Work with your neighbors to define an **interface** for a game of tic-tac-toe. What variables do you need? What methods would you make?
Object-Oriented Programming

The implementation details of an object is defined in a **class** definition. You can think of this as the ‘blueprints’ to make an object.

```c
PyObject * PyString_FromStringAndSize(const char *str, Py_ssize_t size)
{
    ...
    op = (PyStringObject *)PyObject_MALLOC(PyStringObject_SIZE + size);
    if (op == NULL)
        return PyErr_NoMemory();
    PyObject_INIT_VAR(op, &PyString_Type, size);
    op->ob_shash = -1;
    op->ob_sstate = SSTATE_NOT_INTERNED;
    if (str != NULL)
        Py_MEMCPY(op->ob_sval, str, size);
    op->ob_sval[size] = '\0';
    /* share short strings */
    if (size == 0) {
        PyObject *t = (PyObject *)op;
        PyString_InternInPlace(&t);
        op = (PyStringObject *)t;
        nullstring = op;
        Py_INCREF(op);
    } else if (size == 1 && str != NULL) {
        Py_MEMCPY(op->ob_sval, str, size);
        op->ob_sval[size] = '\0';
    }
}
```

This is C code! Don’t worry if you can’t read this.
Python Class Definition

Let’s breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self,r, c, x, y):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

    def __eq__(self, other):
        return (self is other)

    def circumference(self):
        return 2 * Circle.pi * self.radius

    def area(self):
        return Circle.pi * (self.radius)**2
Let's breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self, r, c, x, y):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

    def __eq__(self, other):
        return (self is other)

    def circumference(self):
        return 2 * Circle.pi * self.radius

    def area(self):
        return Circle.pi * (self.radius)**2
Python Class Definition

Let’s breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self, r, c, x, y):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

    def __eq__(self, other):
        return (self is other)

    def circumference(self):
        return 2 * Circle.pi * self.radius

    def area(self):
        return Circle.pi * (self.radius)**2

Let's breakdown a Python class definition:
Given a constructor, we can create new instances of objects.

class Circle:
    pi = 3.14

    def __init__(self, r, c, x, y):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

c1 = Circle(2, "Red", 5, 5)
c2 = Circle(2, "Blue", 5, 10)
c3 = Circle(2, "Red", 5, 5)
Let’s breakdown a Python class definition:

```python
class Circle:
    pi = 3.14

    def __init__(self, r, x, y):
        self.radius = r
        self.color = "Black"
        self.xpos, self.ypos = x, y

    def __init__(self, r, c, x, y):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

    c = Circle(5, 5, 5)
    c = Circle(r=5, x=5, y=5)
```
Let’s breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self, r, x, y, c="Black"):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

c = Circle(5, 5, 5)
c = Circle(r=5, x=5, y=5)
c1 = Circle(2, 5, 5, "Red")
c2 = Circle(2, 5, 10, c="Blue")
c3 = Circle(2, 5, 5, "Red")
Let’s breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self, r, x, y, c="Black"):  
        self.radius = r  
        self.color = c  
        self.xpos, self.ypos = x, y

    def __eq__(self, other):  
        return (self is other)

    def circumference(self):  
        return 2 * Circle.pi * self.radius

    def area(self):  
        return Circle.pi * (self.radius)**2
Python Class Definition

Let’s breakdown a Python class definition:

class Circle:
    pi = 3.14

    def __init__(self, r, x, y, c="Black"):
        self.radius = r
        self.color = c
        self.xpos, self.ypos = x, y

    def __eq__(self, other):
        return (self is other)

    def circumference(self):
        return 2 * Circle.pi * self.radius

    def area(self):
        return Circle.pi * (self.radius)**2

Let’s breakdown a Python class definition:

c1 = Circle(2, "Red", 5, 5)
c2 = Circle(2, "Blue", 5, 10)
c3 = Circle(2, "Red", 5, 5)
print(c1.radius == c2.radius)
print(c1.color == c3.color)
print(c1.area())
print(c1 is c3)
print(Circle.pi)
Interface vs Class vs Object

Unfortunately these are not consistent terms (especially interface)

For this class:

An **interface** is the expected input / output of an object

“A string will always have a length though how I access it might change”

A **class** is the implementation details / coding of the interface

“The Python string class has a method \_\_str\_\(\) which defines print()”

An **object** is a specific instance of a class, with unique variables.

“X and Y are two separate strings with different memory addresses and values.”
In-Class Exercise

How many instances of a list do we have in the following code?

```python
x = [1, 2, 3]
y = x
z = [1, 2, 3]
print(x is y)
print(x is z)
print(x == y)
print(x == z)
print(id(x))
print(id(y))
print(id(z))
```
In-Class Exercise

Let’s return to tic-tac-toe. Can we create a Python class for a common interface?