ICS Spring 2017

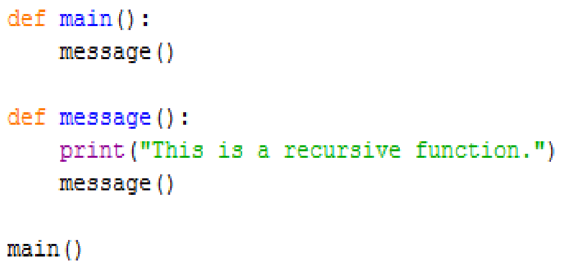
Lab Session Week 5

Lab Agenda:

- Recursion
- Comprehension
- Pickling (Serialization) https://docs.python.org/3.5/library/pickle.html
- OOP

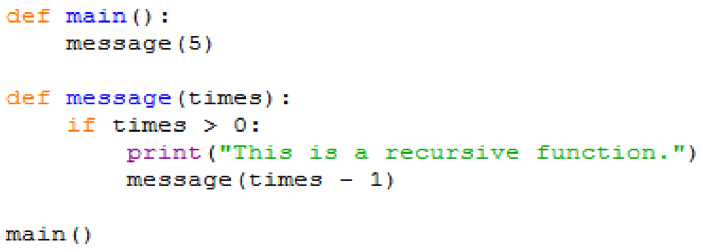
Tuples

- Concept:
  - A recursive function is a function that calls itself.
  - The program below is a recursive function which can run forever like an infinite loop if you don't interrupt with Ctrl+C.



```
def main():
    message()

def message():
    print("This is a recursive function.")
    message()

main()
```

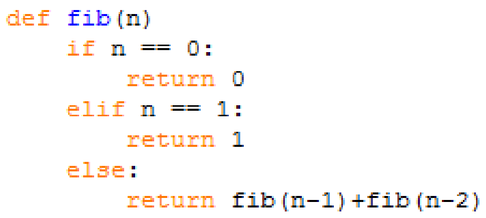
- Similar to a loop, a recursive function can be controlled by some decision structure.
- The if-statement here controls the repetition. The recursion goes on until the argument `times` is not greater than 0, which is determined by times-1 after every recursive call.
- If there are no more statements for recursive structure to execute, the function returns



```
def main():
    message(5)

def message(times):
    if times > 0:
        print("This is a recursive function.")
        message(times - 1)

main()
```

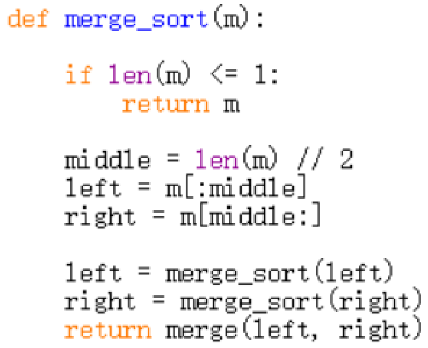
Recursion Examples:

Fibonacci Series:



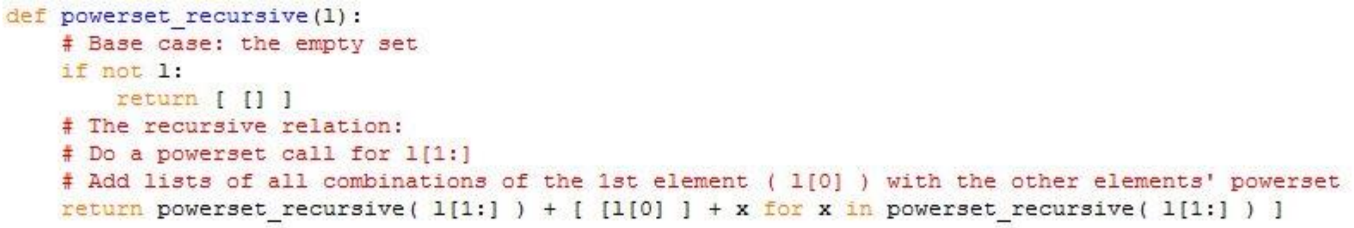
```
def fib(n)
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

Mergesort:



```
def merge_sort(m):

    if len(m) <= 1:
        return m

    middle = len(m) // 2
    left = m[:middle]
    right = m[middle:]

    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)
```

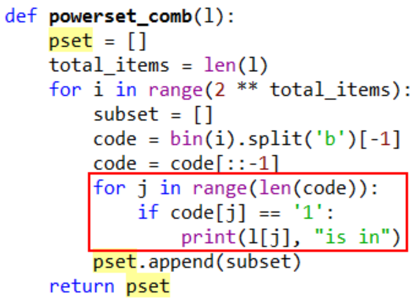
Powerset: also with list comprehension! (Very neat, isn’t it?)



```
def powerset_recursive(l):
    # Base case: the empty set
    if not l:
        return [ [] ]
    # The recursive relation:
    # Do a powerset call for l[1:]
    # Add lists of all combinations of the 1st element ( l[0] ) with the other elements' powerset
    return powerset_recursive( l[1:] ) + [ [l[0] ] + x for x in powerset_recursive( l[1:] ) ]
```

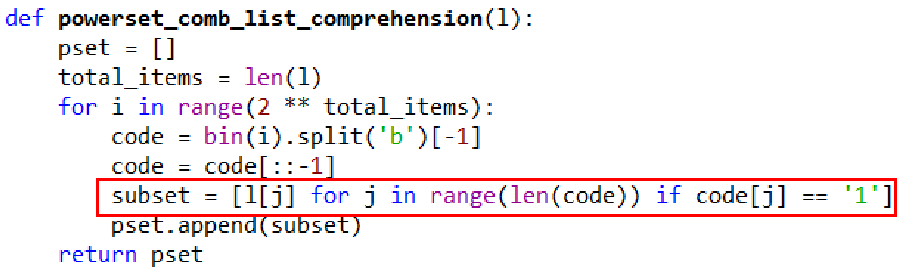
Comprehension

- You can combine lines of code into one single line
  - Looks more precise
  - Looks more advanced
  - Also trains you more with your programming skills
- Example:
  - Before comprehension:



```
def powerset_comb(l):
    pset = []
    total_items = len(l)
    for i in range(2 ** total_items):
        subset = []
        code = bin(i).split('b')[-1]
        code = code[::-1]
        for j in range(len(code)):
            if code[j] == '1':
                print(l[j], "is in")
        pset.append(subset)
    return pset
```

  - After comprehension:



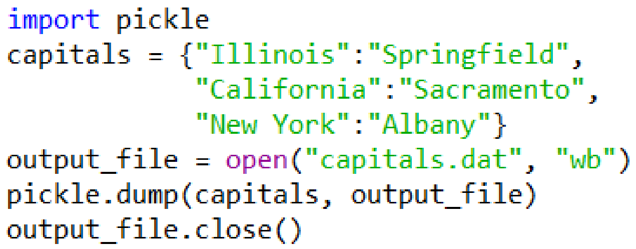
```
def powerset_comb_list_comprehension(l):
    pset = []
    total_items = len(l)
    for i in range(2 ** total_items):
        code = bin(i).split('b')[-1]
        code = code[::-1]
        subset = [l[j] for j in range(len(code)) if code[j] == '1']
        pset.append(subset)
    return pset
```

Pickling

- We can directly work with binary files with the pickle module
- To use: import the pickle module, use the appropriate file mode
    - ‘r’ -> ‘rb’ (read binary)
    - ‘w’ -> ‘wb’
- `pickle.`**`dump`**`()`
    - Writes an object directly to file as its native data type (not a string!)
- `pickle.`**`load`**`()`
    - Reads the first available binary object in the file and returns it

Step-by-step Analysis

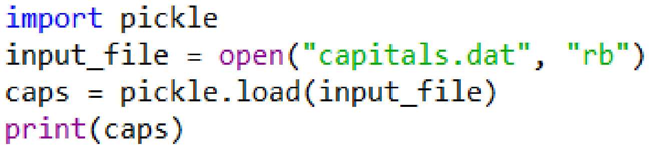
- Create a dictionary containing keys and values
- Open a file for binary writing
- Calls the pickle module’s dump function to serialize the desired dictionary and write it into a dat file
- Close the dat file



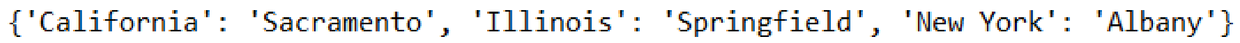
```
import pickle
capitals = {"Illinois":"Springfield",
            "California":"Sacramento",
            "New York":"Albany"}
output_file = open("capitals.dat", "wb")
pickle.dump(capitals, output_file)
output_file.close()
```

Once you have opened a file for binary reading, you call the load function:

- Open your desired dat file for binary reading
- Calls the load function to retrieve and unpickle an object from your dat file.
- You can call the variable in the console and the dictionary will be displayed.
- Better close the file eventually



```
import pickle
input_file = open("capitals.dat", "rb")
caps = pickle.load(input_file)
print(caps)
```



```
{'California': 'Sacramento', 'Illinois': 'Springfield', 'New York': 'Albany'}
```

Object Oriented Programming (OOP)

Self-study tutorials:
https://www.youtube.com/watch?v=ZDa-Z5JzLYM

The whole series takes 6 clips, but we will skip No.5, and for this week, No.1 and No.2 are recommended.

CLASSES

- Time to shift from procedural programming to object oriented programming
  - Defining program flow in terms of **objects** (data + related methods) versus simply telling the interpreter what to do, step-by-step
  - **Encapsulation** of data (variables) and the ways to work with it (methods/functions)
- Advantages: Reusability, Organization, Modularity

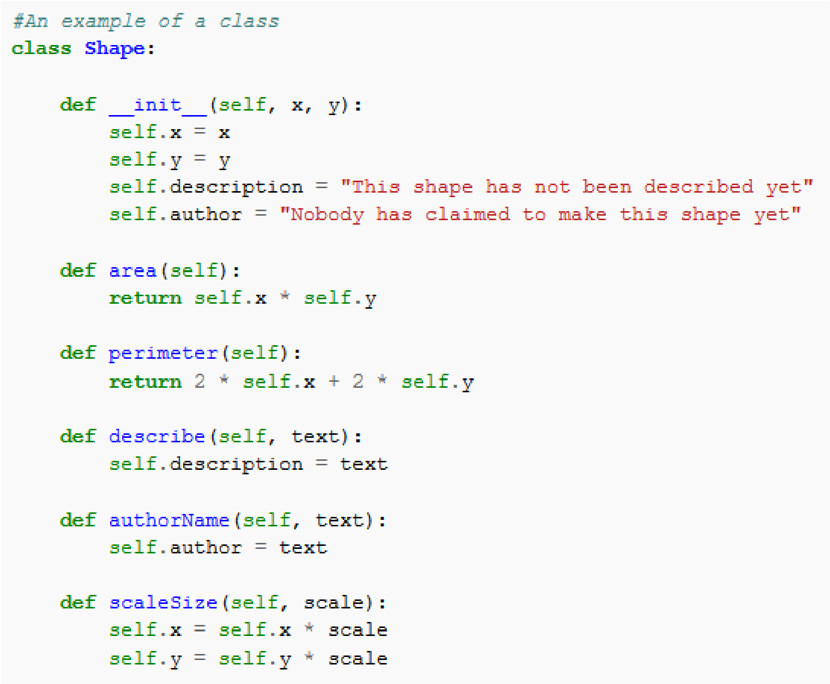
Concept of Class and Object

- “Class” refers to a blueprint. It defines the variables and methods the objects support
- “Object” is an instance of a class. Each object has a class which defines its data and behavior

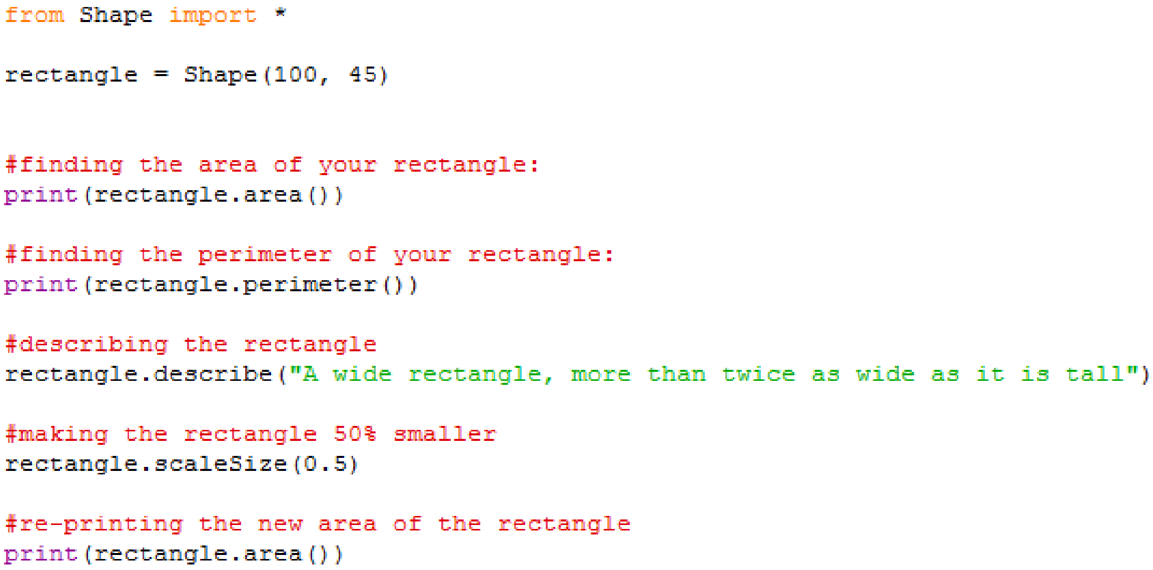
- Making a class
  - The **Class** keyword
  - The blueprint (class) vs. the instance (object)
- Important class methods:
  - **__init__():** a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
  - You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.

Class Members

- **A class can have three kinds of members:**
  - ***fields***: data variables which determine the status of the class or an object
  - ***methods***: executable code of the class built from statements. It allows us to manipulate/change the status of an object or access the value of the data member
  - ***nested classes and nested interfaces***



```
#An example of a class
class Shape:

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.description = "This shape has not been described yet"
        self.author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y

    def describe(self, text):
        self.description = text

    def authorName(self, text):
        self.author = text

    def scaleSize(self, scale):
        self.x = self.x * scale
        self.y = self.y * scale
```



```
from Shape import *

rectangle = Shape(100, 45)


#finding the area of your rectangle:
print(rectangle.area())

#finding the perimeter of your rectangle:
print(rectangle.perimeter())

#describing the rectangle
rectangle.describe("A wide rectangle, more than twice as wide as it is tall")

#making the rectangle 50% smaller
rectangle.scaleSize(0.5)

#re-printing the new area of the rectangle
print(rectangle.area())
```