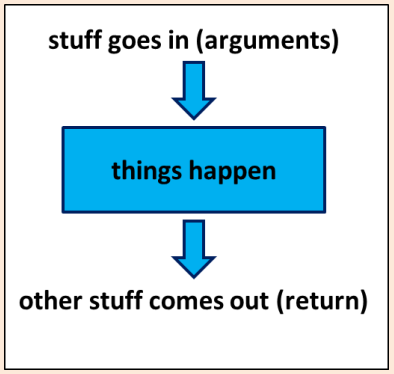
Functions and Modules

Genome 559: Introduction to Statistical and Computational Genomics

Elhanan Borenstein

A quick review

- **Functions**:
  - Reusable pieces of code (write once, use many)
  - Take arguments, "do stuff", and (usually) return a value
  - Use to organize & clarify your code, reduce code duplication



- Defining a function:

```
def <function_name>(<arguments>):
    <function code block>
    <usually return something>
```

- Using (calling) a function:

```
<function defined here>

<my_variable> = function_name(<my_arguments>)
```

A quick review

- Functions have their own namespace
  - Local variables inside the function are invisible outside
- Arguments can be of any type!
  - Number and strings
  - Lists and dictionaries
- Return values can be of any type!
  - Number and strings
  - Lists (as a way to return multiple values)
- **Pass-by-reference vs. pass-by-value**
- Default arguments

Modules

Modules

- Recall your makeDict function:

```
def makeDict(fileName):
    myFile = open(fileName, "r")
    myDict = {}
    for line in myFile:
        fields = line.strip().split("\t")
        myDict[fields[0]] = float(fields[1])
    myFile.close()
    return myDict
```

- This is in fact a very useful function which you may want to use in many programs!
- So are other functions you wrote (e.g., makeMatrix)

Modules

- A module is a file that contains a collection of **related** functions.
- You have already used several built-in modules:
  - e.g.: sys, math
- Python has numerous standard modules
  - Python Standard Library: (http://docs.python.org/library/)

- **It is easy to create and use your own modules:**
  - **JUST PUT YOUR FUNCTIONS IN A SEPARATE FILE!**

Importing Modules

- To use a module, you first have to import it into your namespace
- To import the entire module:

```
import module_name
```

my_prog.py

```
import utils
import sys

Dict1 = utils.makeDict(sys.argv[1])
Dict2 = utils.makeDict(sys.argv[2])

Mtrx = utils.makeMatrix("blsm.txt")

…
```

utils.py

```
# This function makes a dictionary
def makeDict(fileName):
    myFile = open(fileName, "r")
    myDict = {}
    for line in myFile:
        fields = line.strip().split("\t")
        myDict[fields[0]] = float(fields[1])
    myFile.close()
    return myDict

# This function reads a 2D matrix
def makeMatrix(fileName):
    < ... >
```

The dot notation

- Why did we use `utils.makeDict()` instead of just `makeDict()`?

- Dot notation allows the Python interpreter to organize and divide the namespace

Code like a pro …

Code like a pro …

Write comments!

Why comments

- **Uncommented code = useless code**
- **Comments are your way to communicate with:**
  - Future you!
  - The poor bastard that inherits your code
  - Your users (most academic code is open source!)
- **At minimum, write a comment to explain:**
  - Each function: target, arguments, return value
  - Each File: purpose, major revisions
  - Non-trivial code blocks
  - Non-trivial variables
  - Whatever you want future you to remember

Best (real) comments ever

```
# When I wrote this, only God and I understood what I was doing
# Now, God only knows
```

```
# I dedicate all this code, all my work, to my wife, Darlene,
# who will have to support me and our three children and the
# dog once it gets released into the public.
```

```
# I am not responsible of this code.
# They made me write it, against my will.
```

```
# drunk. fix later
```

```
# Magic. Do not touch.
```

```
# I am not sure if we need this, but too scared to delete.
```

```
# Dear future me. Please forgive me.
# I can't even begin to express how sorry I am.
```

```
# no comments for you!
# it was hard to write so it should be hard to read
```

```
# somedev1 - 6/7/02 Adding temporary tracking of Logic screen
# somedev2 - 5/22/07 Temporary my ass
```

Sample problem #1

- Write a function that calculates the first n elements of the Fibonacci sequence.
  - Reminder: In the Fibonacci sequence of numbers, each number is the sum of the previous two numbers, starting with 0 and 1. This sequence begins: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, …
- The function should return these n elements as a list

Solution #1

```
# Calculate Fibonacci series up to n
def fibonacci(n):
    fib_seq = [0, 1];
    for i in range(2,n):
        fib_seq.append(fib_seq[i-1] + fib_seq[i-2])

    return fib_seq[0:n]    # Why not just fib_seq?

print fibonacci(10)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

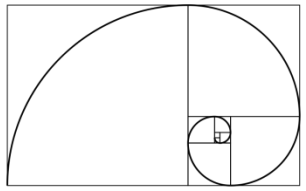
Sample problem #2

- Make the following improvements to your function:

1. Add two **optional** arguments that will denote alternative starting values (instead of 0 and 1).
   - fibonacci(10) → [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
   - fibonacci(10,4) → [4, 1, 5, 6, 11, 17, 28, 45, 73, 118]
   - fibonacci(10,4,7) →[4, 7, 11, 18, 29, 47, 76, 123, 199, 322]

2. Return, in addition to the sequence, also the ratio of the last two elements you calculated (how would you return it?).

3. **Create a module "my_math" and include your function in this module. Import this module into another program and use the function.**



Solution #2

my_math.py

```
# Calculate Fibonacci series up to n
def fibonacci(n, start1=0, start2=1):
    fib_seq = [start1, start2];
    for i in range(2,n):
        fib_seq.append(fib_seq[i-1]+fib_seq[i-2])

    ratio = float(fib_seq[n-1])/float(fib_seq[n-2])
    return [fib_seq[0:n], ratio]
```

my_prog.py

```
import my_math
seq, ratio = my_math.fibonacci(1000)
print "first 10 elements:",seq[0:10]
print "ratio:", ratio
# Will print:
# first 10 elements: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
# ratio: 1.61803398875

fib = my_math.fibonacci # creating a local name
print fib(5,12,14)
# Will print:
# [[12, 14, 26, 40, 66], 1.65]
```

Challenge problem

- Write your own sort function!
- Sort elements in ascending order.
- The function should sort the input list **in-place**
  (i.e. do not return a new sorted list as a return value; the list that is passed to the function should itself be sorted after the function is called).
- As a return value, the function should return the number of elements that were in their appropriate ("sorted") location in the original list.
- You can use any sorting algorithm. Don't worry about efficiency right now.

Challenge solution 1

```
def swap(a_list, k, l):
        temp = a_list[k]
        a_list[k] = a_list[l]
        a_list[l] = temp

def bubbleSort(a_list):
    n = len(a_list)
    a_list_copy = [] # note: why don't we use assignment
    for item in a_list: a_list_copy.append(item)

    # bubble sort
    for i in range(n):
        for j in range(n-1):
            if a_list[j] > a_list[j+1]:
                swap(a_list, j, j+1) # note: in place swapping

    # check how many are in the right place
    count = 0
    for i in range(n):
        if a_list[i] == a_list_copy[i]: count += 1
    return count
```

This is the actual sorting algorithm. Simple!

```
>>> ls = [1, 3, 2, 15, 7, 4, 8, 12]
>>> print bubbleSort(ls)
2
>>> print ls
[1, 2, 3, 4, 7, 8, 12, 15]
```

Challenge solution 1

```
def swap(a_list, k, l):
        temp = a_list[k]
        a_list[k] = a_list[l]
        a_list[l] = temp

def bubbleSort(a_list):
    n = len(a_list)
    a_list_copy = [] # note: why don't we use assignment
    for item in a_list: a_list_copy.append(item)

    # bubble sort
    for i in range(n):
        for j in range(n-1-i):
            if a_list[j] > a_list[j+1]:
                swap(a_list, j, j+1) # note: in place swapping

    # check how many are in the right place
    count = 0
    for i in range(n):
        if a_list[i] == a_list_copy[i]: count += 1
    return count
```

Why is this better?
Why is this working?

```
>>> ls = [1, 3, 2, 15, 7, 4, 8, 12]
>>> print bubbleSort(ls)
2
>>> print ls
[1, 2, 3, 4, 7, 8, 12, 15]
```