Functions

Genome 559: Introduction to Statistical and Computational Genomics

Elhanan Borenstein

A quick review

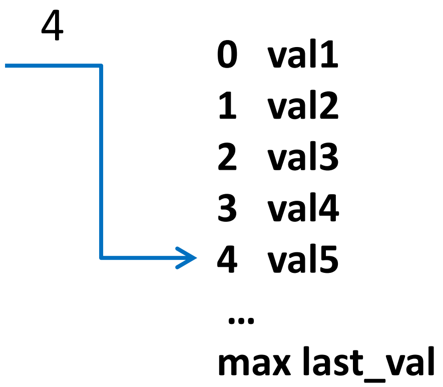
- Dictionaries:
  - key:value pairs
  - a.k.a. hash tables, lookup tables
  - Examples:
    - Word and definition
    - Name and phone number
    - Gene name and score
    - Username and password
  - Dictionaries are useful when you want to look up some data (value) based on a key
  - Each key can appear only once
- Standard I/O

Note: dictionary and list access times

- Accessing a list by index is very fast!
- Accessing a dictionary by key is very fast!

- Accessing a list by value (e.g. list.index(myVal) or list.count(myVal)) can be SLOW.

by index:

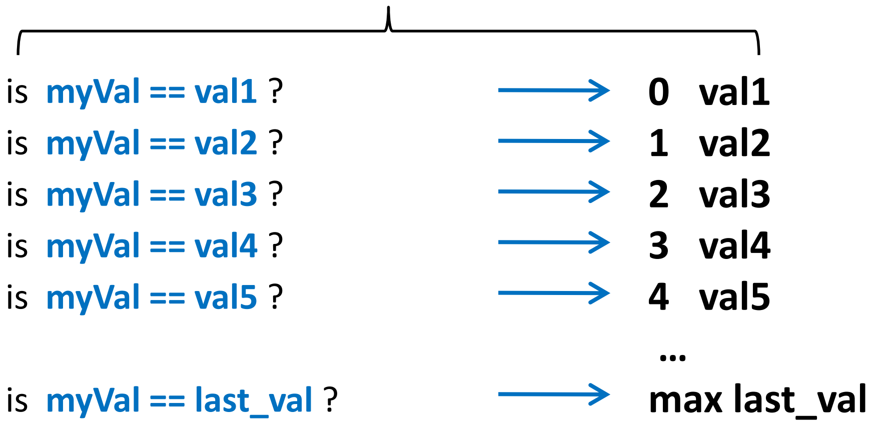


4

0 val1
1 val2
2 val3
3 val4
4 val5
…
max last_val

(index points directly to position in memory)

by value:



is myVal == val1 ? → 0 val1
is myVal == val2 ? → 1 val2
is myVal == val3 ? → 2 val3
is myVal == val4 ? → 3 val4
is myVal == val5 ? → 4 val5
…
is myVal == last_val ? → max last_val

Take a deep breath …

… and think how much you've learned!
4 weeks ago, this would have been gibberish:

```
import sys
matrixFile = open(sys.argv[1], "r")
matrix = []                              # initialize empty matrix
line = matrixFile.readline().strip()     # read first line stripped
while len(line) > 0:                     # until end of file
    fields = line.split("\t")            # split line on tabs, giving a list of strings
    intList = []                         # create an int list to fill
    for field in fields:                 # for each field in current line
        intList.append(int(field))       # append the int value of field to intList
    matrix.append(intList)               # after intList is filled, append it to matrix
    line = matrixFile.readline().strip() # read next line and repeat loop
matrixFile.close()

for row in matrix:                       # go through the matrix row by row
    for val in row:                      # go through each value in the row
        print val,                       # print each value without line break
    print ""                             # add a line break after each row
```

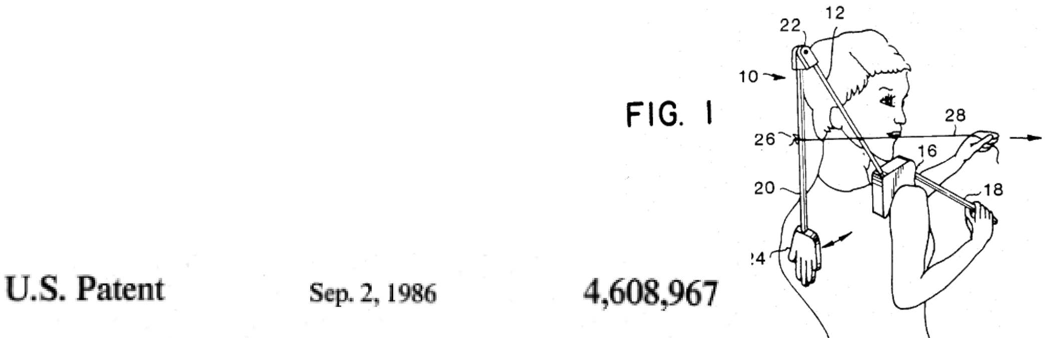

U.S. Patent Sep. 2, 1986 4,608,967

FIG. 1

In theory,
what you know so far allows you to solve any computational task
(“universality”)

So … why don’t we stop here?

most real-life tasks
will be (very) painful to solve
using only what you know so far ...

What are we missing?

What are we missing?

- A way to generalized procedures …
- A way to store and handle complex data …
- A way to organize our code …
- Better design and coding practices …

Functions

Why functions?

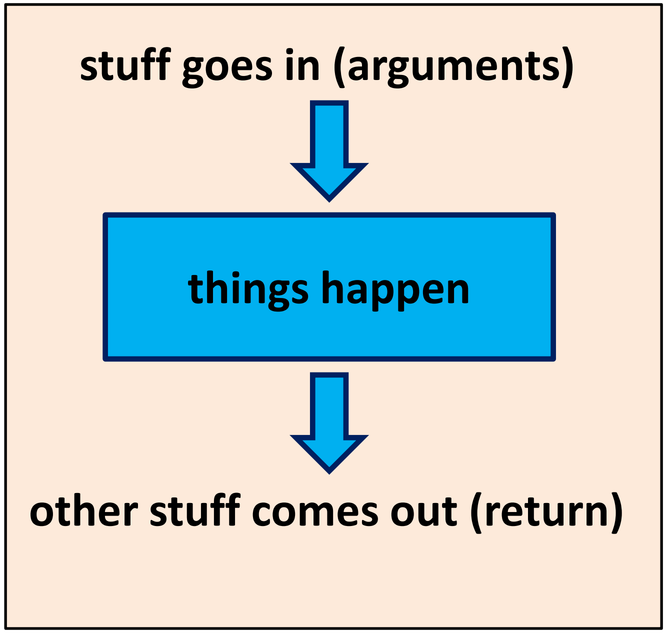
- **Reusable piece of code**
  - write once, use many times
  - Within your code; across several codes

- **Helps simplify and organize your program**

- **Helps avoid duplication of code**

What a function does?

- Takes defined inputs (***arguments***) and may produce a defined output (***return***)



- Other than the arguments and the return, **everything else inside the function is invisible outside the function** (variables assigned, etc.). Black box!
- The function doesn't need to have a return.
- Spoiler: The arguments can be changed and changes are visible outside the function

Jukes-Cantor model

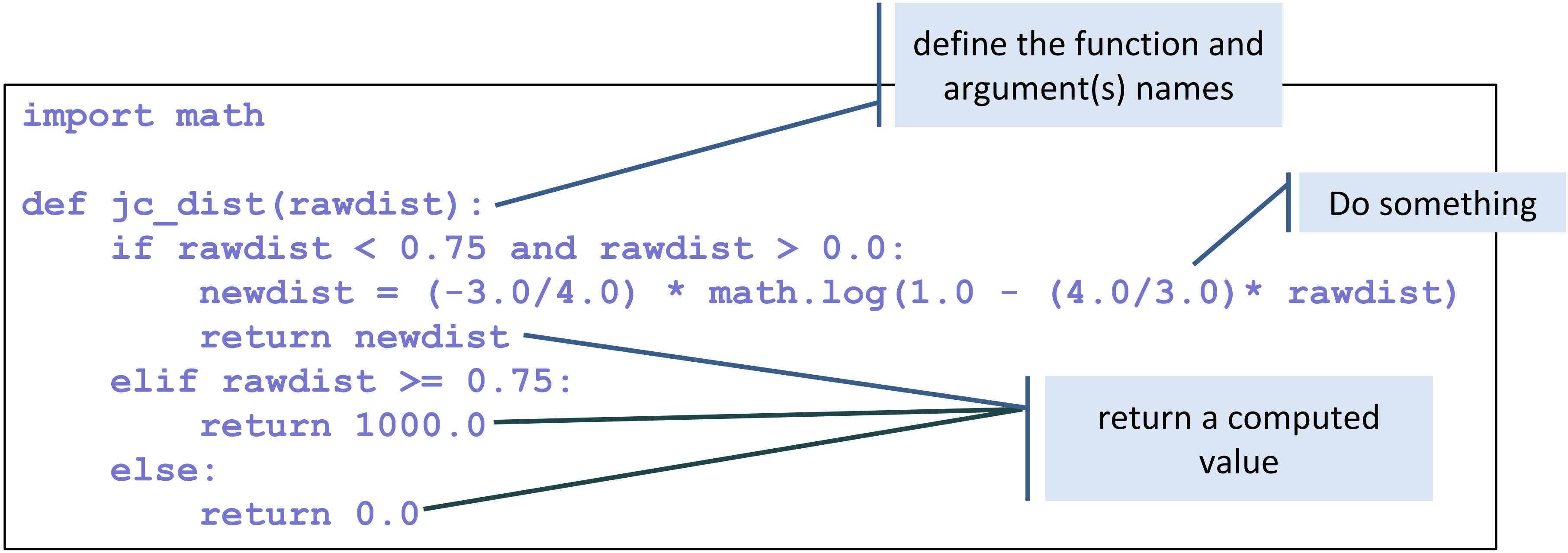
Jukes-Cantor model:

$$D = -\frac{3}{4}\ln(1-\frac{4}{3}D_{raw})$$

D_{raw} is the raw distance (what we directly measure)

D is the corrected distance (what we want)

Defining a function



```
import math

def jc_dist(rawdist):
    if rawdist < 0.75 and rawdist > 0.0:
        newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
        return newdist
    elif rawdist >= 0.75:
        return 1000.0
    else:
        return 0.0
```

define the function and argument(s) names

Do something

return a computed value

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

Using (calling) a function

```
<function defined here>

import sys
dist = sys.argv[1]
correctedDist = jc_dist(dist)
```

Once you've written the function,
you can forget about it and just use it!

Using (calling) a function

```
<function defined here>

import sys
dist = sys.argv[1]
correctedDist = jc_dist(dist)

AnotherDist = 0.354
AnotherCorrectedDist = jc_dist(AnotherDist)

OneMoreCorrectedDist = jc_dist(0.63)
```

Using (calling) a function

```
<function defined here>

import sys
dist = sys.argv[1]
correctedDist = jc_dist(dist)

AnotherDist = 0.354
AnotherCorrectedDist = jc_dist(AnotherDist)

OneMoreCorrectedDist = jc_dist(0.63)

# What about …
jc_dist(0.57)
```

From “In-code” to Function

Jukes-Cantor distance correction written directly in program:

```
import sys
import math

rawdist = float(sys.argv[1])
if rawdist < 0.75 and rawdist > 0.0:
    newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
    print newdist
elif rawdist >= 0.75:
    print 1000.0
else:
    print 0.0
```

Jukes-Cantor distance correction written as a function:

```
import sys
import math

def jc_dist(rawdist):
    ~~rawdist = float(sys.argv[1])~~
    if rawdist < 0.75 and rawdist > 0.0:
        newdist = (-3.0/4.0) * math.log(1.0 - (4.0/3.0)* rawdist)
        return newdist
    elif rawdist >= 0.75:
        return 1000.0
    else:
        return 0.0
```

Add a function definition

delete - use function argument instead of argv

return value rather than printing it

We've used lots of functions before!

```
math.log(value)
readline(), readlines(), read()
sort()
split(), replace(), lower()
```

- These functions are part of the Python programming environment (in other words they are already written for you).
- Note - some of these are functions attached to objects (and called object "methods") rather than stand-alone functions. We'll cover this later.

Function names, access, and usage

- Giving a function an informative name is very important! Long names are fine if needed:

  ```
  def makeDictFromTwoLists(keyList, valueList):
  def translateDNA(dna_seq):
  def getFastaSequences(fileName):
  ```

- For now, your function will have to be defined within your program and before you use it. Later you'll learn how to save a function in a module so that you can load your module and use the function just the way we do for Python modules.

- Usually, potentially reusable parts of your code should be written as functions.

- Your program (outside of functions) will often be very short - largely reading arguments and making output.

Sample problem #1

Below is part of the program from a sample problem. It reads key - value pairs from a tab-delimited file and makes them into a dictionary. Rewrite it so that there is a function called makeDict that takes a file name as an argument and returns the dictionary.

```
import sys
myFile = open(sys.argv[1], "r")
# make an empty dictionary
scoreDict = {}
for line in myFile:
    fields = line.strip().split("\t")
    # record each value with name as key
    scoreDict[fields[0]] = float(fields[1])
myFile.close()
```

Here's what the file contents look like:

```
seq00036<tab>784
seq57157<tab>523
seq58039<tab>517
seq67160<tab>641
seq76732<tab>44
seq83199<tab>440
seq92309<tab>446
etc.
```

Use:

```
scoreDict = makeDict(myFileName)
```

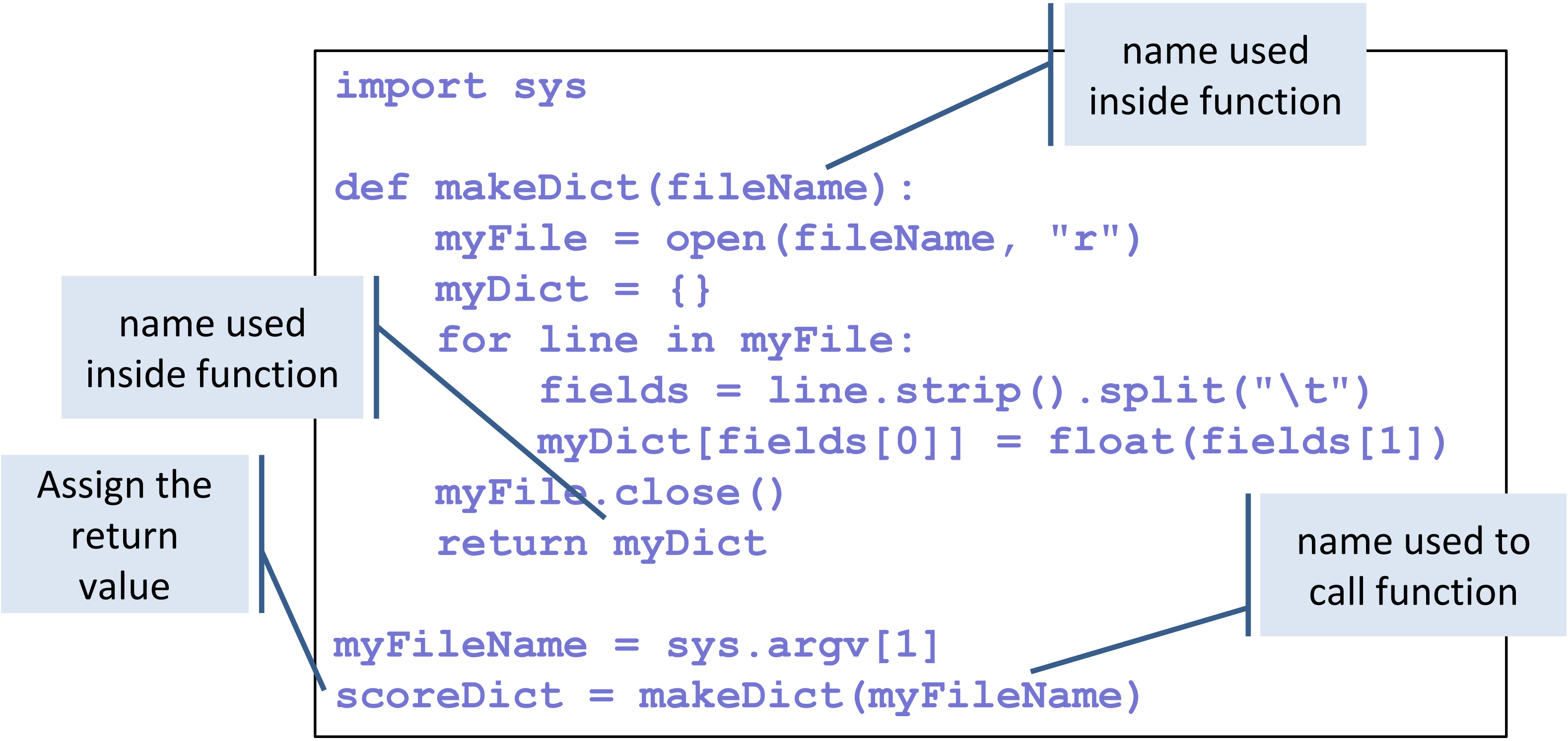
Solution #1

```
import sys

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

myFileName = sys.argv[1]
scoreDict = makeDict(myFileName)
```

Solution #1



```
import sys

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

myFileName = sys.argv[1]
scoreDict = makeDict(myFileName)
```

name used inside function

name used inside function

Assign the return value

name used to call function

Two things to notice here:

- you can use any file name (string) when you call the function
- you can assign any name to the function return

(in programming jargon, the function lives in its own namespace)

Sample problem #2

Write a function that mimics the **<file>.readlines()** method. Your function will have a **file** object as the argument and will return a **list** of strings (in exactly the format of **readlines()**). Use your new function in a program that reads the contents of a file and prints it to the screen.

You can use other file methods within your function, and specifically, the method **read()** - just don't use the **<file>.readlines()** method directly.

Note: This isn't a useful function, since Python developers already did it for you, but the point is that the functions you write are just like the ones we've already been using. BTW you will learn how to attach functions to objects a bit later (things like the split function of strings, as in myString.split()).

Solution #2

```
import sys

def readlines(file):
    text = file.read()
    tempLines = text.split("\n")
    lines = []
    for tempLine in tempLines:
        lines.append(tempLine + "\n")
    return lines

myFile = open(sys.argv[1], "r")
lines = readlines(myFile)
for line in lines:
    print line.strip()
```

Challenge problem

Write a program that reads a file containing a tab-delimited matrix of pairwise distances and puts them into a 2-dimensional list of distances (floats). Have the program accept two additional arguments, which are the names of 2 sequences from the matrix, and print their distance.

Make the matrix reading a function.

Be sure it works with ANY matrix file with this format! The file will always be a square matrix of size (N+1) x (N+1). N for each distance and 1 row and column for names.

Here's what the file contents look like:

```
names<tab>seq1<tab>seq2<tab>seq3
seq1<tab>0<tab>0.1<tab>0.2
seq2<tab>0.1<tab>0<tab>0.3
seq3<tab>0.2<tab>0.3<tab>0
Etc.
…
```

```
>python dist.py matrixFile seq2 seq3
0.3
```

Hints: use the first line to make a dictionary of names to list indices; your function should return a 2-dimensional list of floats.

Challenge solution

I wrote both complex parts as functions; this makes the point that once these are written and debugged, the program is simple and easy to read (the last three lines).

```
import sys

def makeMatrix(fileName):
    myFile = open(fileName, "r")
    myMatrix = []
    lines = myFile.readlines()
    for rowIndex in range(1,len(lines)):
        fields = lines[rowIndex].strip().split("\t")
        matRow = []
        for colIndex in range(1,len(fields)):
            matRow.append(float(fields[colIndex]))
        myMatrix.append(matRow)
    myFile.close()
    return myMatrix

def makeNameMap(fileName):
    myFile = open(fileName, "r")
    line = myFile.readline();
    myFile.close()
    nameMap = {}
    fields = line.strip().split("\t")
    for index in range(1,len(fields)):
        nameMap[fields[index]] = index - 1
    return nameMap

distMatrix = makeMatrix(sys.argv[1])
nameMap = makeNameMap(sys.argv[1])
print distMatrix[nameMap[sys.argv[2]]][nameMap[sys.argv[3]]]
```

(this could be done more efficiently - this way you open the file twice)

looks up the argument string as the key in nameMap, which returns the index of the name in the 2-dimensional list of distance values