Fundamental Python for bioinformatics

BY DAVID E. HUFNAGEL

DEVELOPED BASED ON BCBGSO PYTHON WORKSHOPS

CREDIT TO YUAN WANG

Why program?

- Automation brings speed and reproducibility, and removes the risk of human error
  - The dangers of copy/pasting and working in GUI editors
- “If you’re doing it manually, you’re doing it wrong”

Why Python

- Python is a high-level language (much simpler, also slower)
- Easy to read, write, and edit
- Many libraries make it powerful
- Designed to be fun and straight-forward

Python 2

- I will be teaching Python 2 because
  - I am very experienced with Python 2
  - Python 2.6 and 2.7 are still widely used and many of the best improvements in Python 3 have been backported to Python 2.6 and 2.7
  - Default versions on most Linux and Mac computers are Python 2
  - Python 2 and 3 have few differences, and you can easily move from using one too the other at a later date if you so choose

Tools you will need

- A computer
- Python itself
  - https://www.python.org/downloads/
- An editor
  - Anaconda and Spyder
    - https://www.anaconda.com/download/
  - IDLE comes with python

Getting started

- To use Python one line at a time just type “`python`” into a UNIX terminal
- As per tradition type ”`print ‘hello world`’”
- Both single and double quotes work fine, just be consistent
- To exit Python type “`exit()`” or “`quit()`”
- The same thing can be done with a script using vim, Spyder, or IDLE

Python syntax

- Indentation: Python uses indentation to indicate blocks of code the number of tabs/spaces in indentation for all statements within the same block needs to be the same.
  - (Show example on board)
- Comments: any line starting with “#” will not be executed
- Like UNIX, Python is case-sensitive
- Also like UNIX, one can simply use the up arrow to get the last command

Python Math

- Every Python object has a data type, or a category it lies in that define what can be done with it.
- Only 2 data numeric types: “int” for integer and “float” for numbers containing decimals
- Ints: 1,5,973
- Floats: 5.2, 3.14159
- Many operators:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo (division remainder)
**	Exponent
//	Floor division

Mind the data type

- Int + int = int
- Int – int = int
- Int / int = int
- Int * int = int

- float + float = float
- float – float = float
- float / float = float
- float * float = float

- Int + float = float
- Int – float = float
- Int / float = float
- Int * float = float

Mind the data type

- Int + int = int
- Int – int = int
- Int / int = int
- Int * int = int

- float + float = float
- float – float = float
- float / float = float
- float * float = float

=

- Int + float = float
- Int – float = float
- Int / float = float
- Int * float = float

Defining variables

- No need to separately define and assign
- Not type specific
- Memory is handled for you
- Any variable can be overwritten with anything else
- “a = 2” ←int
- ”b = 2.1” ← float
- Use the “type(variable)” function to determine the data type of a variable

Defining variables

- Always use a smart name
  - Not a single character
  - Truly descriptive (not myList)
- Variable names
  - cannot start with a number
  - Use camelCase or under_scores

Example 1:Calculate the area of a sphere

```
import math


#Area = 4*pi*r^2
r = 5.2
sphereArea = 4 * math.pi * r**2
print sphereArea
```

Strings

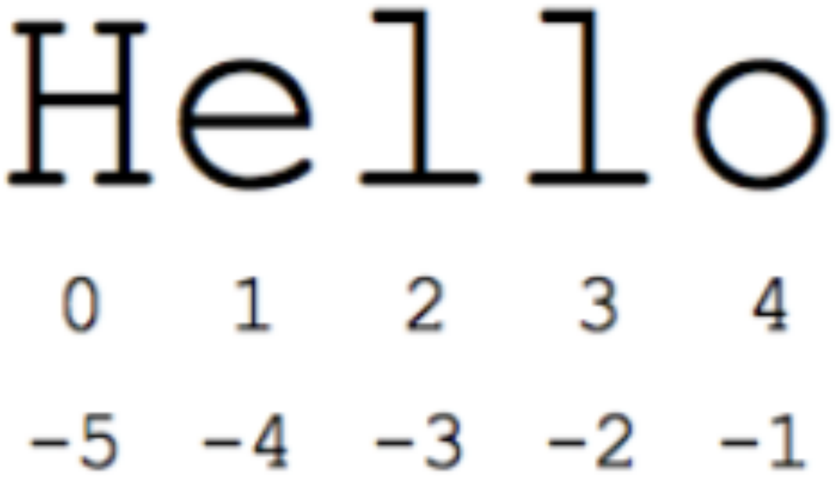
- s = “h”
- s = “hello world”
- s = ’h’
- s = ‘hello world’
- Ordered and immutable

String operations

- “string + string” concatenates two strings into one
- “string * int” creates a new string that is a repeat of the original string with “int” number of repeats

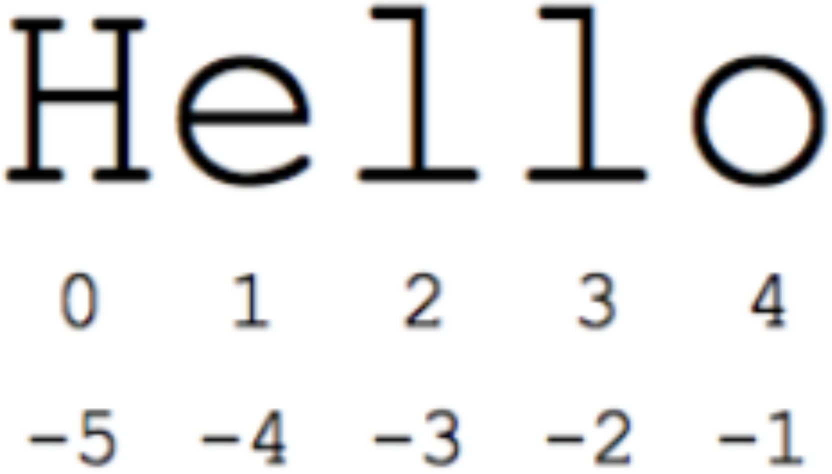
String Indexing

- One character can be extracted from a string using the syntax: string[index]
- Indexes are numbers starting at 0



String slicing

- Multiple characters can be extracted from a string using the syntax: string[startIndex:stopIndex:stepSize]
- When slicing the first number is included, and the last is excluded
  - In Math this is described in this way: [start,stop)



H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

Methods and Functions

- Both are like functions in math, they take 0 or more inputs, do work using the inputs, and output something
- Very similar in Python
- The difference is not important, but they have a structural difference
  - Method: input.method(parameterA, parameterB)
  - Function: method(input, parameterA, parameterB)

If you get stuck/lost

- www.python.org
  - Tutorial
  - Library Reference
  - Language Reference
- https://wiki.python.org/moin/BeginnersGuide/Programmers
  - Very many tutorials
- Google and stackoverflow.com

Fasta format

```
>SeqA

AAGATCTCGAACTAGGCATCAT

>SeqB

ATCGACTGCTAAAGTGCTAGTCCTGAT
```

String Methods

- startswith, endswith, index, replace, count

Method	Description
s.startswith(”string”)	tests if the string object starts with a string argument
s.endswith(“string”)	tests if the string object ends with a sring argument
s.index(“string”)	Searches the string object for a string argument. Returns the index of the string argument. Fails if the string argument is not present.
s.replace(“oldStr”,”newStr”)	returns the string object with all occurrences of 'oldStr' have been replaced by 'newStr'
s.count(“string”)	Counts all occurrences of a string argument in the string object

String Methods

- “\t” = tab
- “\n” = end of line

s.strip(“string")	Removes a string argument from the start and end of the string object. With no string argument specified, removes characters like space, tab, and end of line
s.split(“string”)	Splits the string object into pieces based on the given string argument, returning a list

The Only Important String Function

- “len(string)” returns the length of the string, meaning the number of characters it contains (including letters, numbers, special characters, spaces etc.)

Converting between data types

str(item)	Convert item to a string type. Item can be an int or a float
int(item)	Convert item to an int. Item can be a float or a string representing a number
float(item)	Convert item to a float. Item can be an int or a string representing a number

Example 2

- Lets count GC content in a sequence

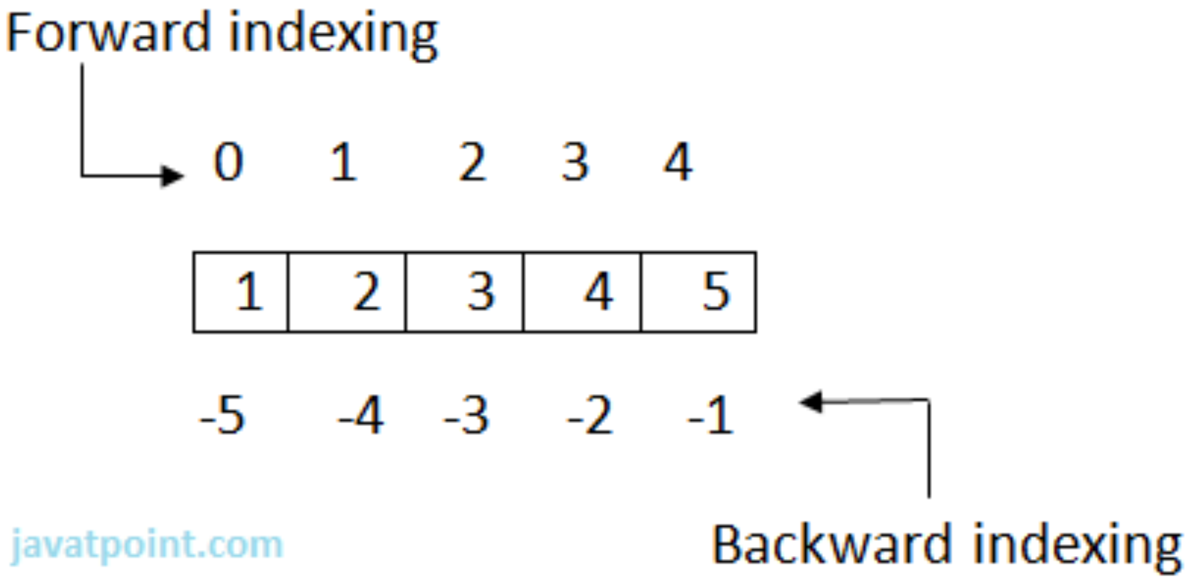
```
seq = “ATCCAGAATCTTA”

Gcount = seq.count(“G”)

Ccount = seq.count(“C”)

gcContent = (Gcount + Ccount) / float(len(seq))

print gcContent
```

Lists

- A list is a collection of objects in a particular order
- Ordered, mutable, can hold any data type including lists
  - Matrix = list of lists
- Uses the structure, “myList = [“hello”, 4, 3.1, [“my”, ”deeper list”]]

Indexing and Slicing lists

- Indexing and slicing a list is just like with strings
- Remember lists are mutable!



javatpoint.com

Operators and lists

- ”+” can be used to concatenate lists
- “-” is of no use
- “*” can be used to repeat a list, but two lists cannot be multiplied together element by element (no matrix math this way)

Extending lists

- myList.append(item)
- myList.extend(item)
- Both these methods add another item to the list
- The difference comes when said item is another list
  - Append makes the whole list item the new last element of the list object
  - Extend adds each item within the list item to the object list

List Methods

- myList.sort() sorts the list from smallest to largest values when containing ints or floats. For strings its more/less alphabetical.

myList.count(item)	returns how many times object item occurs in myList
myList.index(item)	returns the index of the first occurrence of item in myList
myList.insert(index, item)	inserts item into myList at a particular index
myList.remove(item)	removes the first occurrence of item in myList
string.join(myList)	converts the list into a string by concatenating all objects in the list (must already be strings) with string as a separator

List Functions

len(myList)	Returns the number of items in the list
min(myList)	Returns the smallest value item in the list (all items in the list must be ints or floats for a predictable result)
max(myList)	Returns the largest value item in the list (all items in the list must be ints or floats for a predictable result)
sum(myList)	Returns the sum of all value items in the list (all items in the list must be ints or floats)

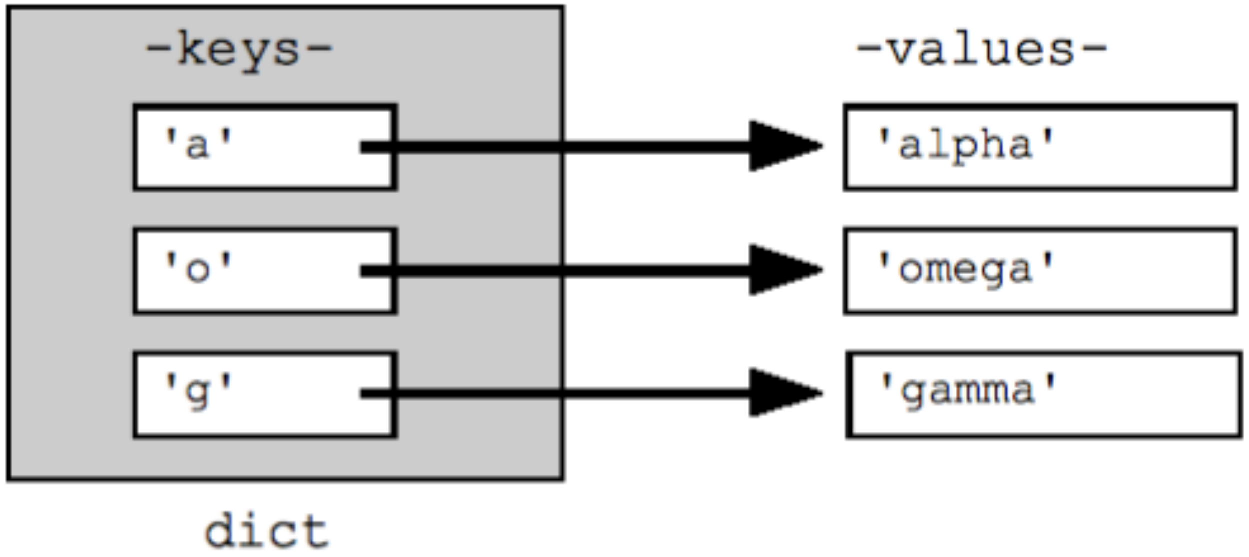
Example 3

- Create a list of lists of genes and their sizes sorted from small to large

```
geneList = []
geneList.append([552, “geneA”])
geneList.append([1499, “geneB”])
geneList.append([1160, “geneC”])
geneList.append([478, “geneD”])
geneList.sort()
print geneList
```

Dictionaries

- A dictionary is a collection of objects of any kind (including dictionaries) structured as a collection of keys and values
- Mutable, unordered



- “dictX = {}”
- “dictX = {’a’:’alpha’, ‘o’:’omega’, ‘g’:’gamma’}”

Dictionaries

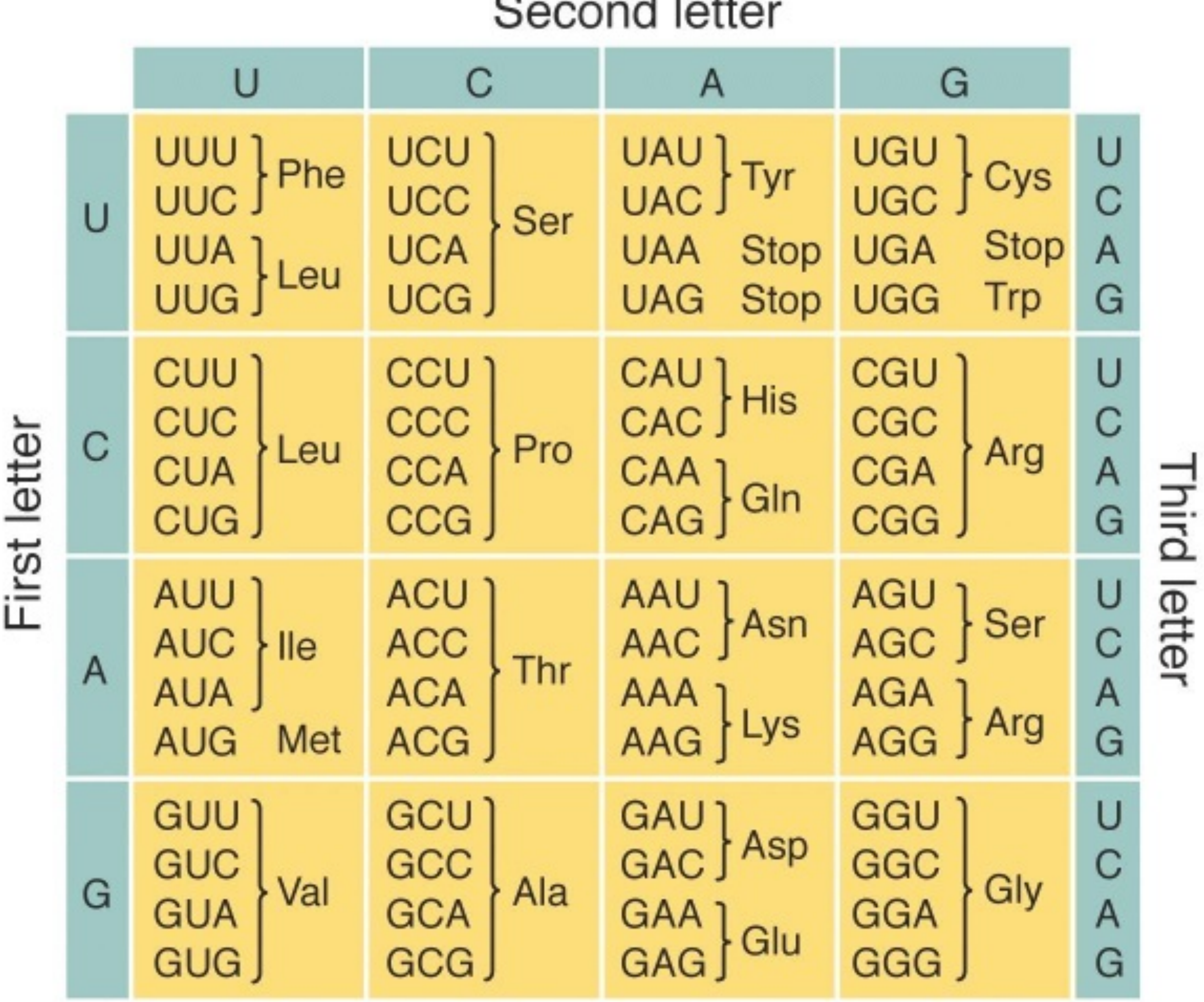
- To add items to an empty dictionary or update values use this structure:
  - dictX = {}
  - dictX[key] = value
- To extract values from a dictionary use this structure:
  - dictX[key]
- Dictionaries are awesome!!

Dictionary methods and functions

del dictX[key]	Removes key and it’s value from dictx
dictX.update(dictY)	Adds all of dictY’s key-value pairs to dictX
len(dictX)	Returns the number of keys in dictx

Translation

- DNA → RNA → Protein
  - Trascription and translation

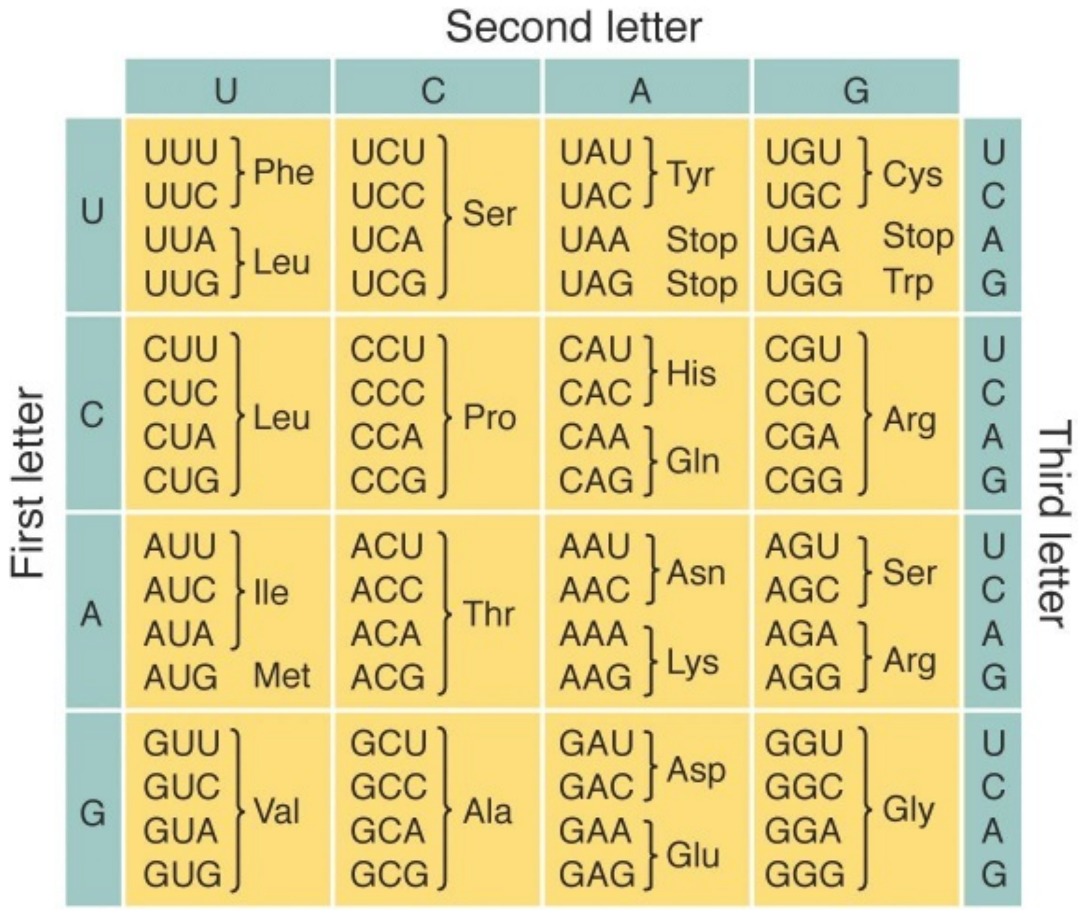


	Second letter								
First letter	U		C		A		G		Third letter
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	Stop	UGA	Stop	A
	UUG		UCG		UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

Example 4

- Create a dictionary to do 1 codon translation

```
translator = {}
translator[“UUU”] = “Phe”
translator[“UUC”] = “Phe”
translator[“UUA”] = “Leu”
translator[“UUG”] = “Leu”
translator[“UAA”] = “STOP”
print translator
```



	Second letter								
First letter	U		C		A		G		Third letter
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	Stop	UGA	Stop	A
	UUG		UCG		UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

Booleans

- A datatype which can be either “True” or “False” (note the capital “T” and “F”)
- False is numerically zero and True is numerically anything nonzero
- Can get a Boolean from a comparison using logical operators

Logical Operators

Operator	Description
==	Equal to
!=	Not equal to
< , > , <= , >=	Less than, greater than, less than or equal to, greater than or equal to
or	Boolean OR
and	Boolean AND
in	Membership test
not	Boolean NOT

Boolean Logic

- True and True → True
- False and False → False
- True and False → False
- False and True → False
- Conclusion: Its true when both A and B are true

Boolean Logic

- True or True → True
- False or False → False
- True or False → True
- False or True → True
- Conclusion: It's true when either A or B is true

Boolean Logic

- True and True and True and True = True
- Anything else with “and” is False (one False makes it False)
- It’s like asking “are all of these True?”

- False or False or False or False = False
- Anything else with “or” is True (one True makes it True)
- It’s like asking “are any of these True?”

Parentheses

- They can be used to group parts of Boolean logic or math

```
((True and True) or False) → True
(True and (True or False)) → True
((3 + 5) * 6) → 48
(3 + (5 * 6)) → 33
```

Conditionals

- Conditionals are used to make decisions about what the script should do based on some or all of the possible conditions
- It’s best to always determine what are all the possible conditions so you can decide what to do about them

```
if condition:
	if_statement
else:
	else_statement
```

- In Python indentation matters, but Spyder and IDLE will both do this for you
- Use a single tab for indentation

Conditionals

```
if condition:
      if_statement
elif condition:
      elif_statement
else:
      else_statement

continued normal python code…
```

Conditionals

```
if condition:
      if_statement
elif condition:
      elif_statement
else:
      else_statement
```

← One Block of code

```
continued normal python code…
```

Conditionals

```
If condition1:
      statement1
elif condition2:
      statement2
else:
      statement3
```

vs.

```
if condition1:
      statement1:
if condition2:
      statement2
else:
      statement3
```

vs.

```
if condition1:
      statement1
elif condition2:
      statement2
elif condition3:
      statement3
```

Conditionals

```
If condition1:
      statement1
elif condition2:
      statement2
else:
      statement3
```

vs.

```
if condition1:
      statement1:
```

```
if condition2:
      statement2
else:
      statement3
```

vs.

```
if condition1:
      statement1
elif condition2:
      statement2
elif condition3:
      statement3
```

Example/Exercise 5

- Generally speaking, proteins start with methionine. Determine if a given sequence starts with the methionine codon, “AUG”

```
seqA = “AGGAAUCNACG”

seqB = “AUGUUAACANN”

…
```

Example 5

- Generally speaking, proteins start with methionine. Determine if a given sequence starts with the methionine codon, “AUG”

```
seqA = “AGGAAUCNACG”
seqB = “AUGUUAACANN”
if seqA.startswith(“AUG”):
      print seqA, “ is a protein!”
else:
      print seqA ” is not a protein”
if seqB.startswith(“AUG”):
      print seqB, “ is a protein!”
else:
      print seqB, ” is not a protein”
```

Best coding Practices

- Comment Frequently
- Use descriptive variable names
- Always plan (Algorithm First)
- No repetition
- Test frequently (for now 1 line at a time)

Algorithms

- An algorithm is a like a recipe, it’s a list of instructions in common language that resembles the structure of a program.

```
seqA = “AGGAAUCNACG”
seqB = “AUGUUAACANN”
if seqA.startswith(“AUG”)
      print seqA, “ is a protein!”
else:
      print seqA ” is not a protein”
if seqB.startswith(“AUG”)
      print seqB, “ is a protein!”
else:
      print seqB, ” is not a protein”
```

Algorithms

1. Define sequences
2. For each sequence
    1. If the sequence starts with methionine print that it is a protein
    2. Otherwise print that it is not a protein

```
seqA = “AGGAAUCNACG”
seqB = “AUGUUAACANN”
if seqA.startswith(“AUG”)
      print seqA, “ is a protein!”
else:
      print seqA ” is not a protein”
if seqB.startswith(“AUG”)
      print seqB, “ is a protein!”
else:
      print seqB, ” is not a protein”
```

For Loops

- Used to do something to every item in a sequence (strings, lists, dictionaries) or to repeat a task many times

```
for char in “Charles Darwin”:
      print char
```

For Loops

- Used to do something to every item in a sequence (strings, lists, dictionaries) or to repeat a task many times

```
for num in [1,2,3]:
      twice = num * 2
      print twice
```

For Loops

- Used to do something to every item in a sequence (strings, lists, dictionaries) or to repeat a task many times
- The “range” function is run this way: `range(start, stop, step)`
  - Like in slicing start is included and stop is excluded

```
print range(8)


for i in range(0,8):
      print i
```

For Loops

- Used to do something to every item in a sequence (strings, lists, dictionaries) or to repeat a task many times
- The “range” function is run this way: `range(start, stop, step)`
  - Like in slicing start is included and stop is excluded

```
for i in range(0,8,2):
      print i
```

Break and Continue

- Sometimes you want to exit the loop when a certain condition is true

```
for i in range(10):
      if i < 7:
            continue
      else:
            break

      print i
      print “so far so good…\n”
```

Break and Continue

- Sometimes you want to exit the loop when a certain condition is true

```
for i in range(10):
      if i < 7:
            continue
      else:
            break

      print i                    ← 1,2,3,4,5,6
      print “so far so good…\n”
```

The “;”

- The “;” can be used to run multiple commands per line (just like in UNIX)

Example/Exercise 6a

- If the sequence is a protein, translate all it’s codons to amino acids (ignore stop codons)
- Create the algorithm…

Example/Exercise 6a

- If the sequence is a protein, translate all it’s codons to amino acids
- Algorithm:

```
define sequences
put sequences in a list
for all sequences
    if they start with Met codon translate all codons to amino acids, and
    store amino acids in a list
print proteins
```

Example/Exercise 6a

- If the sequence is a protein, translate all it’s codons

```
translator = {"UUA":"Leu", "CUG":"Leu", "CGC":"Arg", 
"AGG":"Arg", "AAU":"Asn", "AUG":"Met", "ACA":"Thr", 
"GGG":"Gly"}


seqA = “AGGAAUCUGCGC”;seqB = “AUGUUAACAGGG”

…
```

Example/Exercise 6a

- If the sequence is a protein, translate all it’s codons

```
translator = {"UUA":"Leu", "CUG":"Leu", "CGC":"Arg", "AGG":"Arg",
"AAU":"Asn", "AUG":"Met", "ACA":"Thr", "GGG":"Gly"}

seqA = “AGGAAUCUGCGC”;seqB = “AUGUUAACAGGG”
seqList = [seqA, seqB]
for seq in seqList:
  if seq.startswith(“AUG”):
       allAA = []
       for num in range(0,len(seq),3):
             codon = seq[num:num+3]
             aa = translator[codon]
             allAA.append(aa)
       print allAA
```

Random library

- The library can be reached with ”`import random`”.

random.choice(list/string)	randomly chooses one item from the list/string
random.sample(list/string, k)	Randomly chooses k items from the list/string
random.randrange(start, stop)	randomly chooses an integer between two numbers
random.uniform(start, stop)	randomly chooses a float between two numbers

Example/Exercise 6b

- Generate a random point in a circle with radius 1 (unit radius) and center (0,0) by rejection sampling with 100 iterations. Check “random” functions.

- Algorithm:

1. Set maximum number of iterations
2. For the maximum number of iterations
    1. Randomly generate x coordinate from Unif(-1,1)
    2. Randomly generate y coordinate from Unif(-1,1)
    3. Compute distance from origin (radius)
    4. If radius is less than or equal to 1, end for loop
3. Print coordinates

Example/Exercise 6b

- Generate a random point in a circle with radius 1 (unit radius) and center (0,0) by rejection sampling with 100 iterations. Check “random” functions.

```
import random, math
max_attempts = 100
for i in range(max_attempts):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        r = math.sqrt(x**2 + y**2)
        if r <= 1:
                break
print x, y
```

Inputs and Outputs

- To open a file for reading (input):

```
myFile = open(“inputFileName.txt”)
```

- To open a file for writing (output):

```
myFile = open(“outputFileName.txt”, “w”)
```

- To close a file (any kind)

```
myFile.close()
```

Example/Exercise 7

- Convert fasta to a tab separated 2 column of sequence names and sequence lengths
- Make an algorithm first (I will not check this one)
- fasta file = ”/ptmp/bcbio444/UNIX_Exercises/4_Ex4/TAIR10_peps.mod.fa”

Example/Exercise 7

```
fasta = open("TAIR10_peps.mod.fa")
out = open("TAIR10_peps.mod.fa.lengths", "w")
seqLen = 0
for line in fasta:
        if line.startswith(">"):
                if seqLen != 0:
                        out.write(str(seqLen))
                        out.write("\n")
                        seqLen = 0
                name = line.strip().strip(">")
                out.write(name) #\n is included here
                out.write("\t")
        else:
                seq = line.strip()
                seqLen = seqLen + (len(seq))
out.write(str(seqLen))
out.write("\n")
fasta.close();out.close()
```

End of lab

- Good work!
- I’ll see you to complete Python tomorrow

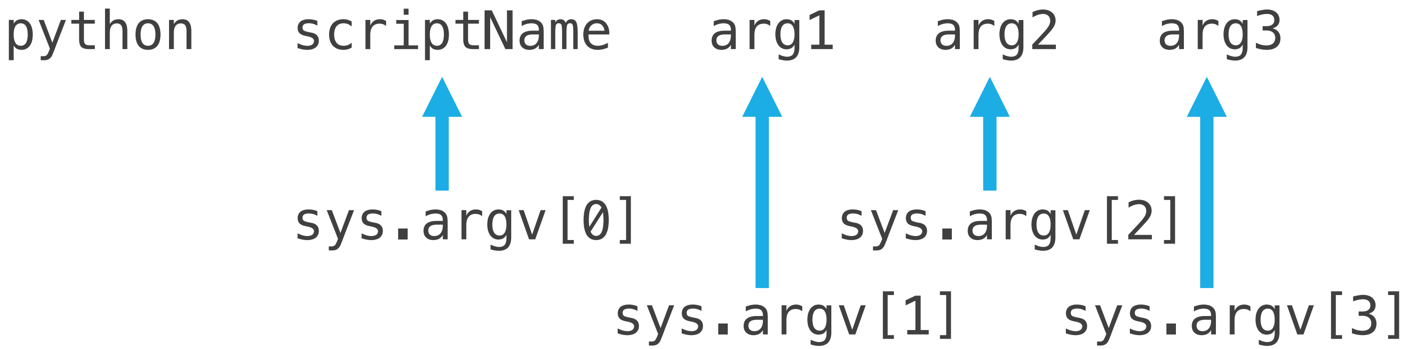
Dynamic Inputs and Outputs

- When working in the terminal it’s nice to be able to have a script that will run on any file it is given, and not just those that are “hardcoded” as in:

```
myFile = open(“inputFileName.txt”)
```

- This can be done using the ”sys” library
- Two functions:
  - Dynamic I/O
  - sys.exit()

Dynamic Inputs and Outputs



```
python    scriptName    arg1    arg2    arg3
```

sys.argv[0]

sys.argv[1]

sys.argv[2]

sys.argv[3]

Dynamic Inputs and Outputs

From Before:

```
myFile = open(“inputFileName.txt”)
```

Instead:

```
import sys
myFile = open(sys.argv[1])
```

and in the terminal:

```
python myScript.py inpA
```

Backslash and the 80 character limit

- There is a Python tradition for the sake of readability
  - The 80 character limit
- “\” can be used to break a line into many pieces

Example 8

- Given any sequence, if the sequence starts with “AUG” translate all the amino acids and print them, otherwise state that it is not a protein coding gene.

```
import sys
translator = {see script}
seq = sys.argv[1]
if seq.startswith(“AUG”):
        allProts = []
        for num in range(0,len(seq),3):
                codon = seq[num:num+3]
                prot = translator[codon]
                allProts.append(prot)
        print allProts
else:
        print ”This is not a protein coding gene.”
```

Functions

- This is the solution for repetition
- You’ve seen many built-in functions:
  - len(), str(), max(), sort(), etc.
- Now you can build your own functions!

```
def FunctionName(parameterA, parameterB…):
      do work
      return object(s)
```

Scope

- Variables defined outside of functions and for loops are global, while those defined within functions or for loops are local and can only be used locally.

```
a = 5


for i in range(10):
      print i
      if i == a:
            break
```

Scope

- Variables defined outside of functions and for loops are global, while those defined within functions or for loops are local and can only be used locally.

```
a = 5


for i in range(10):
      print i        → 1,2,3,4,5
      if i == a:
            break
```

Scope

- Variables defined outside of functions and for loops are global, while those defined within functions or for loops are local and can only be used locally.

```
a = 5


for i in range(10):

      a = 7

      print i         → 1,2,3,4,5,6,7

      if i == a:

            break


print a               → 5
```

Example 9

```
def Average(listx):
        total = float(sum(listx))
        avg = total / float(len(listx))
        return avg

myList = [2, 4, 6]
print Average(myList)
```

Standard Script Structure

Intro (layout algorithm and authorship, import libraries and files)

Define functions

Body (most of the work including outputting)

Conclusion (closing files)

Example 10

- take a pseudogene .bed file as input and determine the total pseudogene content on each chromosome.
- .bed format:

Chromosome start stop name optional

- See script

Best coding Practices

- Comment Frequently
- Use descriptive variable names
- Always plan (Algorithm First)
- No repetition
- Test frequently (for now 1 line at a time)