Who am I?

› Most of my experience is on the molecular and animal modeling side

› I also design computer programs for analyzing biological data, particularly genomic data and sequencing

› Biomedical experience areas
  – Metabolism (molecular and in vivo)
  – Developmental biology (in vivo and informatics)
  – Clinical genetics (in vivo and informatics)
  – CRISPR design (informatics)
  – Cancer immunology (informatics)

› I also work with data security as well
  – I will probably mention “defensive coding” at least once
  – If I haven’t mentioned ways people might steal your passwords or put viruses on your computer by the end of the course, ask for your money back
Class structure

› Ending time is more of a suggestion than a rule
  – I will avoid running past the ending time by any significant amount
  – I build in time for questions and helping students debug code.
    Class length is some function of the number of students, number/complexity of questions, and bugs encountered.

› Ask questions and raise hands when things aren’t working.
  – I am here to help. Helping you learn how to help yourself is even better. I will probably ask you to read out your error messages when stuff fails.
  – If you are unable to get something working, please let me know quickly; there is a good chance that it will need to be working for a subsequent exercise.
Topics

› Review of basic data structures
  – Boolean
  – Integer
  – Float
  – List/array/matrix
  – String
  – Dictionary/hash table

› Accessing and working with objects in python

› Numpy

› Pandas

› Matplotlib

› Scipy
Unstructured/minimally structured data
In the beginning, there was the boolean (bit)...

› Can be 1 or 0, True or False, Yes or No, Alive or Dead, Positive or Negative, etc.

› Can be used in logical statements
  – True AND False = False
  – True OR False = True
  – True XOR True = False
  – NOT True = False

› The most atomic unit of digital information

› Otherwise, kind of boring
Stuff you can do with Booleans

› Strong vs. weak DNA base
› Purine vs. pyrimidine
› True vs. False
› Present vs. absent
› Positive vs. negative
› Dead vs. alive
› Success vs. failure
› FLAG values (collection of Booleans represented by an integer)
  – Which leads us to…
Simple sets of Booleans representing integers

› Anything you can represent with a whole number
  – Lengths
  – Numbered indices
  – Rounded values
  – Counts
  – Base positions

› Anything you need to increment
  – Counters
  – Positions
  – Numbered indices for iterations

› Collections of Boolean values…
Lets have a look at the sam file

Line 1

Query/read name (Unique except for pel & pe2)
Flag *** very important

SRR067577.3006
99
chr14
57829687
60
101M
=
57829772
170
CAATCTATTTAAAGTAATCCCTT...
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII...
XT:A:U NM:i:0 SM:i:37 AM:i:37 X0:i:1 X1:i:0 XM:i:0
XO:i:0 XG:i:0 MD:Z:101
Let's have a look at the sam file

Flag – is a value and tells you about mapping

e.g.  Read is paired        1
       Read is mapped in proper pair  2
       Read is unmapped         4
       Read is first in pair     64
       ........
Let's have a look at the SAM file

Flag – is a value and tells you about mapping

e.g. Read is paired 1
     Read is mapped in proper pair 2
     Read is unmapped 4
     Read is first in pair 64
     .......

Flag value of 66 (read is mapped in proper pair and is first in pair)
Let's have a look at the sam file

Flag – is a value and tells you about mapping

e.g.

Read is paired
Read is mapped in proper pair
Read is unmapped
Read is first in pair

…….

Flag value of 66 (read is mapped in proper pair)
Let's have a look at the SAM file.

Flag – is a value and tells you about mapping:

- Read is paired: 1
- Read mapped in proper pair: 2
- Read unmapped: 4
- Mate unmapped: 8
- Read reverse strand: 16
- Mate reverse strand: 32
- First in pair: 64
- Second in pair: 128
- Not primary alignment: 256
- Read fails platform/vendor quality checks: 512
- Read is PCR or optical duplicate: 1024
- Supplementary alignment: 2048

Flag value of 66 (read is mapped in proper pair and is first in pair).

Summary:
- Read paired
- Read mapped in proper pair
- Mate reverse strand
- First in pair
1, 2, 4... 8?? 16?!

Learn how to freakin' count!
supplementary alignment
read is PCR or optical duplicate
two reads fail platform/vendor quality checks
not primary alignment
first in pair
mate reverse strand
mate unmapped
read unmapped
read mapped in proper pair
read paired
Floating-point numbers

› Some standard conventions on how to represent in binary
  – We won’t worry about those here
› Useful for representing rational numbers.
Structured data
Indexed (arrays/lists/matrices)

› May be heterogeneous or homogeneous (depending on language)
› Depending on storage method, can be highly efficient to process via prefetching (pulling data on to the CPU before requested)
› Can be indexed by position number
› Useful for things like time-series, series of letters/numbers/symbols
› A list of lists can be thought of as forming a 2-dimensional matrix
› Python will not allow for reading and/or writing out of bounds; other languages might
  – Reading out of bounds is almost always a very bad thing to do; writing out of bounds is even worse
› Series of characters from text are special kinds of lists/arrays called…
Strings

- A one-dimensional array of characters
  - Can be indexed or sliced by number
  - Almost always denoted by being surrounded in quotes
  - Due to restricted data element types, can allow some extra functionality (changing capitalization, comparing and searching text)
  - Python will not implicitly treat “11” (string value) as the number 11.
    - i.e. “11” + “11” = “1111”; 11 + 11 = 22; “11” + 11 = ☹️ (TypeError)
    - str(11) = “11”; int(“11”) = 11; str(“11”) = “11”; int(11) = 11
    - str(11) + “11” = “1111”; int(“11”) + 11 = 22
    - Other languages may implicitly try to convert
Key:Value (dictionary/hash table)

› Allows the recall of some data by looking up its key
  – Keys must be unique and immutable (hashable)
    › Can’t use things like lists or other dictionaries as a key
  – Values can be any data type, even other key:value sets
  – Entering a new key:value for a key that already exists in the dictionary will replace the existing value

› Has no inherent order for key:value pairs

› Enforced uniqueness of keys can be used to create a set of unique values

› Some languages may not remember the set of keys (Python always does)

› Useful for matching Subject:Result, Gene:mRNA length, Species:Gene:mRNA:Sequence (dictionary of dictionaries of dictionaries of strings)
Objects and object oriented programming (OOP)
Topics

› Review of basic data structures
› Accessing and working with objects in python
   – What are objects?
   – Classes and instances of objects
   – Spoiler alert: You’ve been using objects this whole time
   – Example of a simple object
   – Analyzing a SAM file: Your first object oriented program

› Numpy
› Pandas
› Matplotlib
› Scipy
What are objects

› Combination of functions (called methods) and data (called attributes)

› A Class is a type of object. An object is an instance of its class.
  – TRPV4 and GPIHBP1 are both members of the Gene class
  – Both have very different sequence attributes
  – Both have very different protein function attributes
  – Both may or may not have CG content values, but the method for calculating CG contents would be identical

› Class names start with a capital letter, instance names start with lowercase
  – Good coding practice by convention, not strictly enforced by Python
  – Don’t get in the habit of ignoring this, or you will see it strictly enforced by me

› Methods are called by object.method(arguments)

› Attributes are called by object.attribute
The class
Some instances

\[
\text{stupidHatMeeseeks} = \text{Meeseeks}(\text{hat} = \text{hat.stupid}, \text{hair} = \text{None})
\]
Some instances

\[
\text{stupidHairMeeseeks} = \text{Meeseeks}(\text{hat} = \text{None}, \text{hair} = \text{hair.tuft})
\]
Every data element in Python is an object

- `myNucList.sort()` from the first Python workshop was a method call

```python
>>> myNucList
['A', 'G', 'A', 'C', 'T', 'C']
>>> myNucList.sort()
>>> myNucList
['A', 'A', 'C', 'C', 'G', 'T']
>>> myNucList.remove('G')
>>> myNucList
['A', 'A', 'C', 'C', 'T']
>>> del myNucList[3]
>>> myNucList
['A', 'A', 'C', 'T']
>>> myNucList.pop(1)
'A'
>>> myNucList
['A', 'C', 'T']
```

3 ways to delete
General structure of a class

```python
>>> class MyClass(object):
...     def __init__(self, myRequiredData, myOptionalData = None):
...         self.requiredData = myRequiredData
...         self.optionalData = myOptionalData
...     def combineData(self):
...         if self.optionalData:
...             return self.requiredData + str(self.optionalData)
...         else:
...             return self.requiredData
...     def __str__(self):
...         return self.combineData()
... >>>
>>> test1 = MyClass("something")
>>> test1.combineData()
'something'
>>> str(test1)
'something'
>>> test2 = MyClass("thing1", "thing2")
>>> test2.combineData()
'thing1thing2'
>>> str(test2)
'thing1thing2'
>>> test1.optionalData
>>> test1.requiredData
'something'
>>> test1.myOptionalData
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'MyClass' object has no attribute 'myOptionalData'
```
You may have already mastered the most difficult part...

...git actually has a simple design, with stable and reasonably well-documented data structures. In fact, I'm a huge proponent of designing your code around the data, rather than the other way around, and I think it's one of the reasons git has been fairly successful [...] I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

Torvalds, Linus (2006-06-27). Message to Git mailing list
Also difficult: Finding Linus Torvalds quotes without profanity
Lets have a look at the sam file

Line 1

SRR067577.3006
99
chr14
57829687
60
101M
=
57829772
170
CAATCTATTTAAAGTAATCCCTT...
IIIHIHIIIIIIIIIIIIIIIIHIHIIHI...
XT:A:U NM:i:0 SM:i:37 AM:i:37 X0:i:1 X1:i:0 XM:i:0
XO:i:0 XG:i:0 MD:Z:101
Let's have a look at the sam file

Line 1

SRR067577.3006
99
chr14
57829687
60
101M
=

CAATCTATTTAAAGTAATCCCTTATGTAGCGTCTGACTAGCTAGCCCCTAGCGATGCCCATGAGAGTACGATGAC

XT:A:U NM:i:0 SM:i:37 AM:i:37 X0:i:1 X1:i:0 XM:i:0
XO:i:0 XG:i:0 MD:Z:101
A DNA sequence as an object

CAATCTATTAAAGTAATCCCCTTATGTAGCGTCTGACTAGCTAGCCCCTAGCGATGCCCATGAGAGTACGATGAC

› Attributes
  – The sequence itself
  – Length (from a method or function)
  – Percent \{A, T, G, C\} (from a method or function)
  – GC content (from a method or function)
  – Reverse complement (from a method or function)

› Methods
  – Subsequences (restriction sites? Promoters? Translation?)
  – Match/mismatch counts and positions
Let's have a look at the sam file

Class SAMLine

Line 1

SRR067577.3006  SAMLine.readln
99  SAMLine.flag  D
chr14  SAMLine.chromosome
57829687  SAMLine.readStartPosition
60  SAMLine.alignmentPhred
101M  SAMLine.cigar
  =  SAMLine.mateChromosome
57829772  SAMLine.mateStartPosition
170  SAMLine.readLength

CAATCTATTTAAAGTAATCCCTT...  SAMLine.sequence

IIIHIHIHIHIHIHIHIHIHIHIHIHIHI...  SAMLine.qualityString

XT:A:U  NM:i:0  SM:i:37  AM:i:37  X0:i:1  X1:i:0  XM:i:0
  XO:i:0  XG:i:0  MD:Z:101
Important question

› How many of these attributes should be classes themselves?
  – We have already shown DNA sequence
  – The flag score could be a class that analyzes the flag score automatically
  – The quality score could be translated to actual numbers
  – Others?

› Time to code!
Create a new file

› pythonClass2/dnaSequenceHandler.py
› You will probably need to create this folder as well
› The name is case sensitive, and this is going to matter a lot in a few minutes, so mind the capital letters
From last class...

```python
# ! /usr/bin/python

def reverseComplement(dnaSequence):
    complementTable = {
        "A": "T",
        "C": "G",
        "G": "C",
        "T": "A",
        "N": "N"
    }

    revComp = ""
    for base in dnaSequence[::-1]:
        revComp += complementTable[base]
    return revComp
```
Not from last class...

But probably your first class
Declare a class and initialize it

class DNASequence(object):
    def __init__(self, sequence):
        self.sequence = sequence.upper()
        self.length = len(sequence)
        self.percentBase = self.getPercentBases()
        self.gcContent = self.getGCContent()
        self.reverseComplement = reverseComplement(self.sequence)
Now that we know what methods we need...

def getPercentBases(self):
    collection = {}
    for base in self.sequence:
        if not base in collection:
            collection[base] = 0
            collection[base] += 1
    for base in collection:
    return collection
...and a few more methods

```python
def getGCContent(self):
    totalPercent = 0
    for base in ["G", "C"]:
        if base in self.percentBase:
            totalPercent += self.percentBase[base]
        else:
            totalPercent += 0  # not absolutely necessary
    return totalPercent

def countMismatches(self, otherSequence):
    assert type(otherSequence) == str, "Error, sequence for comparison must be a string type."
    if not len(self.sequence) == len(otherSequence):
        raise ValueError("Both sequences must be the same length for comparison")
    otherSequence = otherSequence.upper()
    mismatches = 0
    for i in range(len(self.sequence)):
        if self.sequence[i] != otherSequence[i]:
            mismatches += 1
    return mismatches
```
Hint: If you aren’t overloading `__str__`, ask yourself why

def countMismatches(self, otherSequence):
    assert type(otherSequence) == str, "Error, sequence for comparison must be a string type."
    if not len(self.sequence) == len(otherSequence):
        raise ValueError("Both sequences must be the same length for comparison")
    otherSequence = otherSequence.upper()
    mismatches = 0
    for i in range(len(self.sequence)):
        if self.sequence[i] != otherSequence[i]:
            mismatches += 1
    return mismatches

def __str__(self):
    return self.sequence
And some test code

def countMismatches(self, otherSequence):
    assert type(otherSequence) == str, "Error, sequence for comparison must be a string type."
    if not len(self.sequence) == len(otherSequence):
        raise ValueError("Both sequences must be the same length for comparison")
    otherSequence = otherSequence.upper()
    mismatches = 0
    for i in range(len(self.sequence)):
        if self.sequence[i] != otherSequence[i]:
            mismatches += 1
    return mismatches

def __str__(self):
    return self.sequence

if __name__ == "__main__":
    test = DNASequence("ATGCATGCATGC")
    print(test)
    print(test._dict__)
And now to use the test code

Linux and Mac: you will have to use “python3” instead of “python”
Now what?

If your test worked (output looked like mine), save this file in a new folder called pythonClass2 (if you have not already).

Use the filename “dnaSequenceHandler.py” (very case-sensitive, file name is going to be important later)

If the test did not work, hold up your hand. Read off your error message when call on you

If the test did not work for a lot of you, start working with your neighbors to try solving the problem. Reading error messages required, Googling error messages highly encouraged.
How to attack the SAM file

› We know this is a text-based file
› We know the lines are tab-delimited
Pro tip: The python interpreter is your friend

#theMostHashtags
Iterating over the SAM file

```python
def readSAMFileLines(filename):
    lineCollection = []
    samFile = open(filename, 'r')
    line = samFile.readline()
    while line:
        line = line.strip()
        if not line:
            line = samFile.readline()
            continue
        if line.startswith('@'):
            line = samFile.readline()
            continue
        lineCollection.append(SAMLine(line))
    samFile.close()
    print("Read %s lines" %(len(lineCollection)))
    return lineCollection
```
class SAMLine(object):
    def __init__(self, rawLine):
        import dnaSequenceHandler
        self.rawLine = rawLine.strip()
        lineArray = self.rawLine.split("\t")
        self.readID = lineArray[0]
        self.flag = int(lineArray[1])
        self.chromosome = lineArray[2]
        self.readStartPosition = int(lineArray[3])
        self.alignmentPhred = int(lineArray[4])
        self.cigar = lineArray[5]
        self.mateChromosome = lineArray[6]
        if self.mateChromosome == ":"
            self.mateChromosome = self.chromosome
            self.mateStartPosition = int(lineArray[7])
        self.readLength = int(lineArray[8])
        self.sequence = dnaSequenceHandler.DNASequence(lineArray[9])
        self.qualityString = lineArray[10]
        self.otherValues = lineArray[11:]

    def __str__(self):
        return self.rawLine
Almost done

The cat got a cheeseburger.

we can all go home.

The internet is over. Good game, everyone.
A bit of test code

```python
if __name__ == "__main__":
    samLines = readSAMFileLines("sampleData.sam")
    for i in range(10):
        print(samLines[i])
```
Let's try it out
Objects help keep code organized
  – Note how there was a clear hierarchy of how data was being handled and accessed

Objects help keep code simple and short
  – Note how I never had a function I couldn’t easily fit on the screen
  – You probably made some errors along the way. They would have been much harder to find if this was a single block

Objects help keep code reusable
  – Make the best possible object for your data type, import it everywhere and never worry about writing it again

Objects help keep code accessible
  – Coding out how to access a complex data type may be difficult, but calling data = myComplexDataClass(complexData) is easy
  – Calling data.theThingIActuallyCareAbout is also easy
Homework

You have code and you have sample data. Play around with these things today and become more familiar with them.