Day 2

- Lists
- Simple iteration
- Splitting and sorting

Review from last time

- There are 3 basic types in python: *int*, *str*, and *float*
- Each data type has intrinsic operations that can be performed on them, and type-specific interpretations of common operators like '+' and '/'
- Variables can be used to store values, and they take on the properties of the stored value
- Conditionals can be tested in the context of '*if*' statements
- Try the review exercises

Slice notation

- Every character in a string has a position
- “0” is the first position, and characters are numbered sequentially
- To get a single character from a string (either stored as a variable or as a literal string in quotes), the notation is: *str*[position]
  - ex. 'ACG'[0] returns 'A', 'ACG'[1] returns ‘C’
- To get multiple character, the notation is: *str*[x:y] where x is the first character to grab and y is the position after the last character to be grabbed
  - ex. 'ACG'[0:1] returns 'A', ACG'[0:2] returns 'AC’
  - 'ACG'[3] is impossible but 'ACG'[2:3] returns 'G’
  - if x='ACG', x[0:2] x[0], and x[0:1] would also work

Additional slice properties

- Leaving off the first argument in a slice range defaults to position '0’
  - ex. 'ACGT'[:3] returns 'ACG’
- Leaving off the second argument in a slice range defaults to the last position
  - ex. 'ACGT'[1:] returns 'CGT’
- Negative arguments in slice notation count from the end
  - ex. 'ACGT'[-1:] returns 'T’
  - 'ACGT'[:-1] returns 'ACG'

Nested slice notation

- Slices that return “sliceable” things can be sliced again.
  - Ex. 'ACGT'[1:3][0] returns 'C' Try Exercises 4!

Exercises 4 review

- Any portion of a string can be retrieved with slice indices
- If you know you want to exclude the last *n* letters in something, or only retrieve the last *n* letters in something, negative indices can be useful
- Sometimes slice indices can be a pain as you have to count out the exact start and end positions you want.

More complicated variables

- Variables can hold multiple items. A variable that holds a string, for example, is holding multiple characters
- Lists:
  - can hold *int*, *str*, *float*, and even other lists
  - store elements (also called items) in numerical order
  - denoted with square brackets to mark lists and commas to mark elements in the lists
  - can be sliced just like strings to get individual items back or substrings within individual items
  - Ex.
    - list1=['hey', 5, 2.6]
    - list2=[['this', 3, 'ACG'], ['gene2', 45, 'MPQ']]
    - list3=[list1, list2]
    - list3=[['hey', 5, 2.6], [['this', 3, 'ACG'], ['gene2', 45, 'MPQ']]]

Properties of lists

- Lists maintain the order you initially establish for them unless modified
- The ordering of list elements can be modified
- Individual items from lists can be retrieved quickly if you know the position within the list
- Because lists can store lists, infinitely complex datasets can be constructed, all referenced by a single variable
- For small applications, you can test for presence / absence of things very quickly

Splitting things

- Strings can be split into lists using ".split()"
- ".split(*characters*)" splits a string by occurrences of *characters,* and returns a list of the split pieces
- This is extremely useful when reading lines of text from a file that has columns python doesn't know how to interpret
- Ex.
  - example_string='abzcdzefgzhijkzlm'
  - example_list=example_string.split('z')
  - example_list is now ['ab', 'cd', 'efg', 'hijk', 'lm']
  - example_list[1][1] returns 'd'
- To split by tab, the tab character is '\t'
  - Ex. 'hi	this	is	tabbed'.split('\t') gives: ['hi', 'this', 'is', 'tabbed']
- Try exercises 5!

Exercises 5 conclusion

- Python doesn't know 'columns' of data, but you can split text into lists using the column delimiters your eye is used to, and then retrieve the column you want using slice notation (makes grabbing the right slice easier)
- Slice notation called on lists retrieves (lists of) elements of that list, whereas on strings it retrieves substrings
- A list slice that retrieves a string element can be sliced into a substring
- break, then go on to Exercises 6

Exercises 6 conclusion

- The elements in a list can be directly reassigned to other values with slice notation, but the elements in strings cannot be reassigned without overwriting the variable
- If you want part of a string and don't want to count letters, you can use unique phrases and characters to split and grab smaller and smaller pieces
- Complex statements can be used directly as input to methods and functions

Introducing Loops

- Loops allow commands to be done again and again.
- Line before a loop with ':' signals that a loop is starting and how many times to do the commands.
- Looped commands are indented to define which commands are in the loop
- First unindented line after a ':' defines end of loop
- Loops and conditionals can be nested within each other
- In Bioinformatics, loops often go through all parts of something (ex. characters in a string, genes in a list, lines in a file, or files in a folder).

Introducing Loops

- “for-loops”

```
for item in multi_item_thing:
    repeating_command1
    repeating_command2
```

- The loop finishes after the last item in the *multi_item_thing*

Conditional loops

- “while-loops”

```
while condition == True:
    repeating_command1
    repeating_command2
```

Try Exercises 7!

Loops conclusion

- “for-loops” retrieve every character in the context of a string, and every item in the case of a list
- At each iteration the retrieved item or character is stored to a completely arbitrary variable name of your choosing
- If you only want items meeting certain criteria, or particular substrings or elements, you can use conditionals and slice notation