Principles and Applications of Modern DNA Sequencing

EEEB GU4055

Session 2: Python

Today's topics

1. review notebook assignments (Python Intro)
2. objects, types, variables, return, print
3. iterables: strings and lists
4. conditional statements
5. writing functions

Notebook 2.0: Intro to Python

Every language has its idiosyncrasies. Whether you've never seen Python before, or you're more familiar with another programming language like R, it takes some time to become familiar with the format and rules of any specific language and why they matter.

This primer on Python and bash is intended to introduce and explain some of the reasoning behind these concepts. We will continue to reinforce how and why the code is written the way it is throughout the course.

Python objects

Everything in Python is an object. Different *types* of objects have different features associated with them. This can include functions to query or modify aspects of the object, or ways of returning stats or details about it.

Object-oriented languages are designed for this purpose: connecting functions to the objects they are meant to operate on. It is an organizational structure to help users/coders write cleaner code that is easier to use.

Python objects

The main object types in Python can be created in one of two ways: using a shorthand syntax or explicit function call.

```
# Create objects of various types using their type conventions
"a string"
["a", "list", "of", "strings"]
("a", "tuple", "of", "strings")
{"a key": ["and value in a dictionary"]}
```

```
# Or, we can explicitly use the object type function to creating objects
str("Columbia")
list((1, 2, 3, 4, 5))
tuple("apple", "banana", "orange")
dict([("a", 3), ("b", 4), ("c", 5)])
```

Creating variables to store objects

A created object disappears instantly upon creation unless you store it to a *variable*.

```
# Create objects of various types using their type conventions
a = "a string"
b = ["a", "list", "of", "strings"]
c = ("a", "tuple", "of", "strings")
d = {"a key": ["and value in a dictionary"]}
```

```
# Or, we can explicitly name the object type as a function
a = str("Columbia")
b = list((1, 2, 3, 4, 5))
c = tuple("apple", "banana", "orange")
d = dict([("a", 3), ("b", 4), ("c", 5)])
```

String objects

The string object type is used to represent text. It can be created using the *str()* function or by enclosing text in single or double quotes.

```
# Wrap any text in single or double quotes to create a string
a = "a string"
b = 'another string'
c = "A very long string ....................."
DNA = "ACGCAGTCGATGCTAGCTAGCTGACTGATCGTA"
```

More details on single vs double quotes

The reason that both options exist is that it can be useful to use them in conjunction when the string you wish to create actually includes ' or " character in it.

```
# Use double quotes to enclose a string with single quotes in it
sentence1 = "Deren's dog's name is Phylo"

# Use single quotes to enclose a string with double quotes in it
sentence2 = 'Sometimes we call her "Fart-lo"'

# There is also a special triple-quote option for multiline strings
sentence3 = """
This is a long string that is
broken over multiple lines and
stored with newline characters
"""
```

Strings versus Bytes (Python3)

In Python3 (as opposed to the older Python2) a new object type of 'bytes' was introduced. This is very similar to a string, and it a more efficient way to represent text data. In practice, when you read in data from a file it will sometimes be in 'bytes' format. It is easiest to just convert it to a 'string'.

```
# A bytes object looks like a string but with a 'b' at the beginning
a = str('this is a sentence')
b = bytes('this is a sentence')

# print the two objects
print(a)
print(b)
```

```
'this is a sentence'
b'this is a sentence'
```

Integers and Floats

The integer and float object types are used for mathematical operations.

```
# numeric values (ints or float)
a = 0
b = 10
c = 3300.239291
d = 0.0000301
```

Integers and Floats: Challenge

Challenge 1: In a code cell below write three lines of Python code. On line 1 create a new variable called 'y' with the value 30. On line 2 create another new variable 'z' with the value 5.5. On line 3 use the print function to print the value of y / z.
(See Chapter 3 if you need help).

```
y = 30
z = 5.5
print(y / z)
```

```
5.454545454545454
```

Built-in Functions

A function is a program that performs a task. Functions end in parentheses.

Example: the `len()` function returns the length of an object.

```
# Create a string
DNA = "ACTACTACTACTACTACTAC "

# return the length of the string
len(DNA)
```

```
20
```

Built-in Functions

You might be asking, *but I thought functions are always associated with an object in Python?*. You're rigt. For convenience some functions look and act like standalone functions but are actually associated to objects under the hood. Example:

```
# Create a string
DNA = "ACTACTACTACTACTACTAC "

# len() is a shortcut function
print(len(DNA))

# it actually returns the result of a "hidden" function in the string object
print(DNA.__len__())
```

```
20
20
```

Built-in Functions

Example: string objects have functions to operate on strings, such as to format, search, split, or modify the text in many ways.

```
# Create a string
DNA = "ACTACTACTACTACTACTAC"

# access functions for string objects from the string object
DNA.lower()
```

```
"actactactactactactac"
```

Indexing and slicing

Select a subset of values by their position (starting at 0). Think intervals: [0|1|2|3|4]

```
# Select subsets of an object by their position (starting at 0)
DNA = "ACTACTACTACTACTACTAC"
DNA[0]
```

```
"A"
```

```
DNA = "ACTACTACTACTACTACTAC"
DNA[1:5]
```

```
"CTAC"
```

Indexing and slicing

Challenges: Use indexing to return only the first 10 characters of 'dna'; and only the last 5. (See Chapter 3.1.2 if you need help)

```
dna = "ACGCAGACGATTTGATGATGAGCATCGACTAGCTACACAAAGACTCAGGGCATATA"
dna[:10]
```

```
"ACGCAGACGA"
```

```
dna[-5:]
```

```
"ATATA
```

Indexing and slicing

Challenges: (1) Use the split() function to split the dna variable on the characters "CG". (2) Store the returned result of step 1 to a new variable called dnalist. (3) Then use the print function on the dnalist variable to show its contents.

```
# the dna string variable
dna = "ACGCAGACGATTTGATGATGAGCATCGACTAGCTACACAAAGACTCAGGGCATATA"

# call split with the argument "CG" and store results as dnalist
dnalist = dna.split("CG")

# print to show the value of dnalist
print(dnalist)
```

```
['A', 'CAGA', 'ATTTGATGATGAGCAT', 'ACTAGCTACACAAAGACTCAGGGCATATA']
```

Indexing and slicing

Challenge: In the cell below create two new variables, one called fiveprime that contains the first ten 10 elements in dnalist, and another called threeprime that contains the last 10 elements in dnalist.

```
# the dna string variable
dna = "ACGCAGACGATTTGATGATGAGCATCGACTAGCTACACAAAGACTCAGGGCATATA"

# make dna string into a list object
dnalist = list(dna)

# index the first ten items and store as fiveprime
fiveprime = dnalist[:10]

# index last ten items and store as threeprime
threeprime = dnalist[-10:]
```

Indentation and iterables

Indentation in Python has meaning, where *nested* lines are influenced by the less indented lines above them. For example, a *for-loop*.

```
# format: for each item in container of items do x with item
for letter in "aeiou":
    print(letter)
```

```
"a"
"e"
"i"
"o"
"u"
```

Indentation and iterables

Conditional statements act as a query to do something only *if* something is True or False. The special keyword *if* is used here.

```
# for item in container of items do x with item if it's the right kind.
for letter in "aeiou":
    if letter == "a"
        print(letter)
```

```
"a"
```

Conditional statements: Challenge

Challenge: (1) Create a list object of bases; (2) Iterate over the length of dnalist selecting with indexing; (3) query conditional match the value "A"; (4) if "A" replace with lowercase; (5) print.

```
# 1. create a list
dnalist = list("AAACCCGGGTTT")

# 2. iterate over the index of the list
for i in range(len(dnalist)):

    # 3: select each element and ask if it is "A"
    if dnalist[i] == "A":

        # 4. replace matching "A" with lowercase version
        dnalist[i] = dnalist[i].lower()

# 5. print the final modified version of dnalist
print(dnalist)
```

Conditional statements: Challenge

```
dna1 = "AACTCGCTAAAGCCTCGCGGATCGATAAGCTAG"
dna2 = "AAGTCGCTAAAGCAACGCGGAACGATAACCTGG"

# Hint 1: create an integer variable set to 0
diffs = 0

# Hint 2: use the range function and index each object while iterating
for idx in range(len(dna1)):
    dna1_value = dna1[idx]
    dna2_value = dna2[idx]
    if dna1_value != dna2_value:
        diffs += 1

print(diffs)
```

```
6
```

Functions (writing your own)

Functions are used to perform a repeated task. As we said there are many functions available in Python. In addition, *you can write your own* by using `def()`

```
# a function to add 100 to x
def myfunc(x):
    return x + 100

# run the function on an input value (e.g., 200)
myfunc(200)
```

```
300
```

Functions (writing your own)

Functions are used to perform a repeated task. As we said there are many functions available in Python. In addition, *you can write your own* by using `def()`

```
# name the function and the arguments anything you want
def sumfunc(arg1, arg2):
    summed = arg1 + arg2
    return summed

# run the function
myfunc(200, 300)
```

```
500
```

Comments and documentation

```
def base_frequency(string):
    "returns the frequency of A, C, G, and T as a list"
    # create an empty list
    freqs = []

    # get the full length of the input string
    slen = len(string)

    # iterate over the letters A, C, G, and T
    for base in "ACGT":

        # count letter is in string divided by total and append to results
        freqs.append(string.count(base) / slen)

    # return the results list
    return freqs

# test the function
base_frequency("ACACTGATCGACGAGCTAGCTAGCTAGCTGAC")
```

Challenge: Understanding a mystery function

```
def mystery_function(string):
    "no hint on this one"

    # code block 1
    ag = 0
    ct = 0

    # code block 2
    for element in string:
        if element in ["A", "G"]:
            ag += 1
        elif element in ["C", "T"]:
            ct += 1

    # code block 3
    freq_ag = ag / len(string)
    freq_ct = ct / len(string)

    return [freq_ag, freq_ct]
```

Challenge: Understanding a mystery function

```
def mystery_function(string):
    "Takes input string and returns list with frequency of AG, CT"

    # integer counters
    ag = 0
    ct = 0

    # iterate over string recording purine (AG) or pyrimidine (CT)
    for element in string:
        if element in ["A", "G"]:
            ag += 1
        elif element in ["C", "T"]:
            ct += 1

    # calculate frequency from counts divided by total length
    freq_ag = ag / len(string)
    freq_ct = ct / len(string)

    # returns two frequencies in a list
    return [freq_ag, freq_ct]
```

Importing libraries

Import libraries as objects to access all of the functions and objects from these additional libraries.

```
import random

# generate one random number between 0 and 10
random.randint(0, 10)

# return a list of 10 random numbers
[random.randint(0, 10) for i in range(10)]
```

```
[3, 7, 0, 10, 3, 9, 1, 7, 10, 8]
```

Challenge: many ways to accomplish a task

Write a function using 'random' to generate a random sequence of DNA (As, Cs, Gs, and Ts) of a length that is supplied as an argument. It should return the results as a string object. Demonstrate by generating a 20bp sequence.

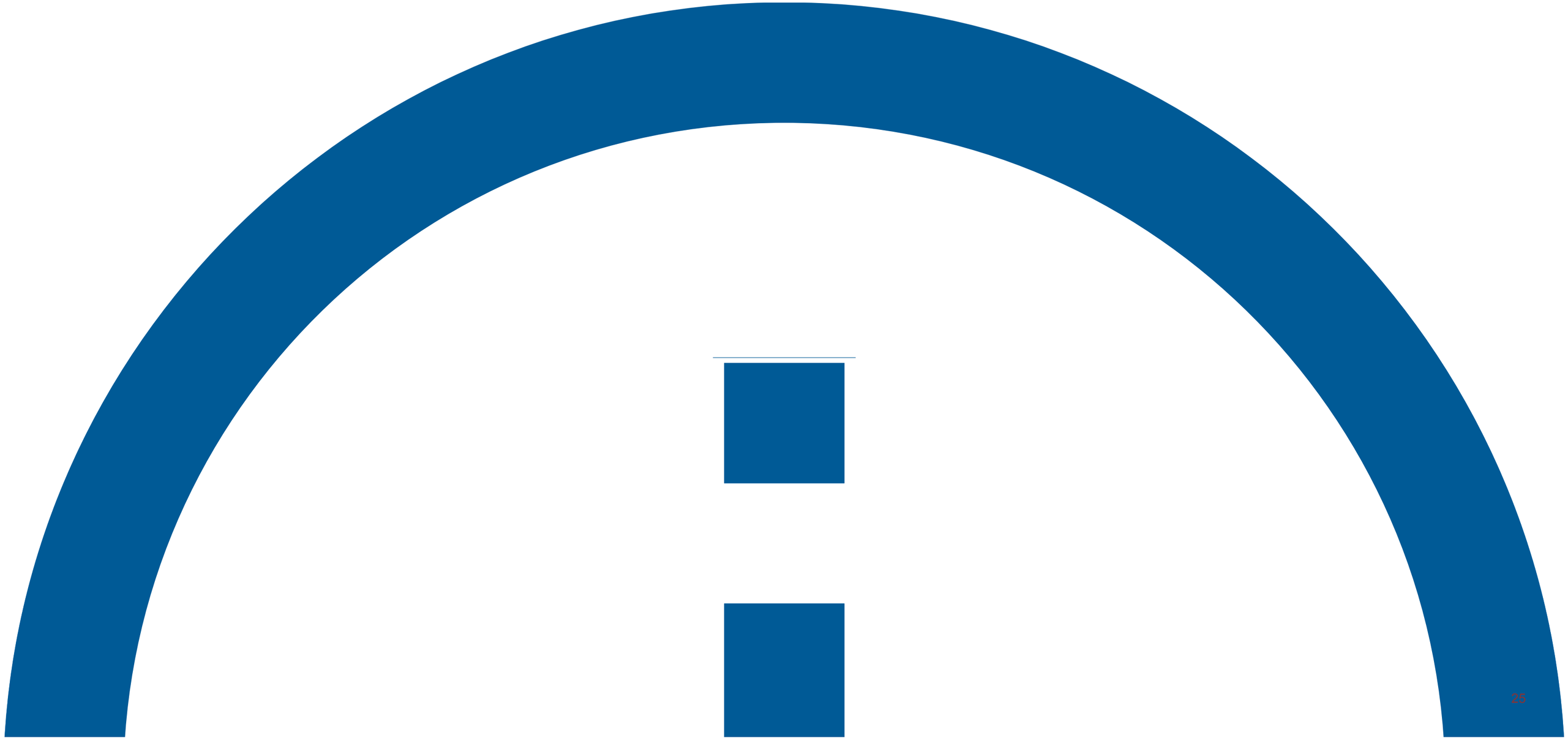
```
def random_dna(length):
    "returns a random string of ACGTs of len length"
    dna = ""
    for i in range(length):
        dna += random.choice("ACGT")
    return dna

# test it out
random_dna(20)
```

```
"AGGTTTTACCGGTATGAGTC"
```

When poll is active, respond at **PollEv.com/dereneaton004**

Post or vote on a question you have about the assignment



Interactive exercise writing functions

Write a function from scratch that uses the 'random' library in some way. An example could be to create a dice rolling function, or a function that randomly mutates a string of DNA that it accepts as an argument. Try to get creative, and share ideas and code with your neighbor.