ace01_python_intro_full

August 21, 2018

0.1 Jupyter Notebook

- The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

Jupyter Notebook https://jupyter-notebook.readthedocs.io/en/stable/

- keyboard shortcuts
  - Help -> Keyboard Shortcuts
- Will learn as we go along...

0.2 Printing and Manipulating Text

In [ ]:
```
# print text to screen - use comments to annotate your code
print("Hello World!")
```

In [ ]:
```
#print() - is the name of a function
#  tells python interpreter what we want to do, in this case print something to the sc
#  the bits inside parentheses are called the arguments, tells interpreter what we wan

# want to know more on print()?
?print()
```

In [ ]:
```
# quotes are important
print("she said: 'Hello, World'")
print('He said: "Hello, World"')
#print("They said: "Hello, World"")
```

In [ ]:
```
# error messages and debugging code
print(Hello World)
# shows: which line error occured
# pythons best guess of where error occured
# the type of error (in this case a syntax error - python can't understand code)

#prin("Hello, World")
# different type of error, NameError.
```

In [ ]:
```
# printing special characters
print("Hello\nWorld") # prints newline character
print("Hello\tWorld") # prints tab
```

In [ ]:
```
# string print formatting
print(5 + 3)
print("5 plus 3: {}".format(5 + 3))
print("5 minus 3: {}".format(5 - 3))
print("5 times 3: {}".format(5 * 3))
print("5 divided by 3: {}".format(5 / 3)) # returns a float
print("5 divided by 3: {:.2f}".format(5 / 3)) # returns a float, 2 decimal places
print("remainder of 5 divided by 3: {}".format(5 % 3))  # modulo, returns the remainde
print("5 cubed: {}".format(5**3))
```

In [ ]:
```
# using variables
x = 27
y = 3

# same math
print("{} divided by {}:  {}".format(x, y, x / y))

# booleans
print("5 less than 3: {}".format(5 < 3))

# boolean in an if loop
if x < y:
    print("{} is greater than {}".format(x, y))
else:
    print("{} is not greater than {}".format(x, y))
```

In [ ]:
```
# storing strings in variables - show tab completion
my_dna = "gagtggatgatgatgggcgctaggcgcgt"
print(my_dna)

print("the object 'my_dna' is of type: {}".format(type(my_dna)))

# everything in python is an object, often having functions available to them
print(my_dna.lower(), my_dna.upper())


# a string can be directly acted on
print("is this lowercase?".upper())

# what methods are available to an object? tab and dir
#print(dir(my_dna))

# can chain methods - dont go overboard!!
print(my_dna.rstrip("gcgcgt").upper())
print(my_dna.upper().rstrip("gcgcgt")) # why doesnt this work?
```

Quick aside - naming conventions https://www.python.org/dev/peps/pep-0008/

In [9]:
```
# example
HomeTown = "Denver" # no for variables
home_town = "Denver"  # yes
```

In [ ]:
```
# manipulating strings - joining
# make upper
my_dna = my_dna.upper()
print(my_dna)

# joining strings
upstream = "atg"
print("upstream concatenated my_dna: {}".format(upstream + my_dna))

# store in variable
new_dna = upstream + my_dna
print("new dna sequence: {}".format(new_dna))

# concatenate multiple
downstream = "tag"
full_dna = upstream + my_dna + downstream
print("full sequence: {}".format(full_dna))

# can join strings in print statement
print("new dna sequence is:" + " " + full_dna)
```

In [ ]:
```
# finding length of a string using len() function
# store length of my_dna
dna_length = len(my_dna)

# want to know more on len()
#?len()


# joining ints and strings
print("length of DNA seq is: " + dna_length)
print("length of DNA seq is: " + str(dna_length))

# or...
print("length of DNA seq is: {}".format(dna_length))
```

In [ ]:
```
# manipulating strings - replacement

protein = "padvlsp"
print(protein)

# replace valine with tyrosine
print(protein.replace("p", "y"))
```

```
        # can replace more than one character - need to be next to each other
        print(protein.replace("pad", "XYZ"))

        # original 'protein' ore variable is not changed
        print(protein)
```

```
In [ ]: # manipulating strings - extracting part of a string - indexing/slicing
        # print first three
        print(protein[1:3])

        # positions start at zero, not one
        # positions are inclusive at the start, and exclusive at the stop
        print(protein[0:3]) # includes the first character and ends before the 4th character

        # print last four
        print(protein[3:]) # not the best
        print(protein[-4:]) # minus sign signals to start from the end
```

```
In [ ]: # finding and counting substrings
        # count number of prolines
        print(dir(protein))
        print(protein.count("p"))

        # store counts
        proline_count = protein.count("p")
        dvl_count = protein.count("dvl")

        # print counts
        print("prolines: " + str(proline_count))
        print("dvl motifs: {}".format(dvl_count))
```

```
In [ ]: # finding location of substring within a string
        #print(dir(protein))
        print(protein.find("p")) # will find the first occurence
        print(protein.find("vls"))
        print(protein.find("fvl")) # -1 indicates not found
```

0.2.1 short intro to files

```
In [ ]: # open a file
        f = open("../data/cities.txt")

        # read contents of a file
        file_contents = f.read()

        # must close
        f.close()
```

```
        # print contents of file
        print(file_contents)
```

```
In [ ]: # write contents out to new file
        # open a new file to write to
        out_file = open("../data/cities_out.txt", "w")
        out_file.write("{}".format(file_contents.replace("o","oi")))
        out_file.close()
```

0.2.2 lists, loops, and dictionaries

```
In [ ]: # lists are most versatile data type in python, written as a list of comma separated v
        # square bracket
        cities = ["houston", "portland", "memphis"]

        # items in list can be accessed by index
        print(cities[1])
        print(cities[1:])

        # can loop through lists
        numbers = [1,2,3,4,5,6,7]
        numbers_squared = []
        for i in numbers:
            #print(i, i**2)
            numbers_squared.append(i**2)

        print(numbers_squared)
```

```
In [ ]: # read in file of locus tags with gene products, populate dictionary
        gene_dict = {}
        with open("../data/gene_names.txt") as f:
            for line in f:
                line = line.rstrip("\n")
                gene_id, gene_name = line.split("\t")
                gene_dict[gene_id] = gene_name

        gene_dict
```

```
In [ ]: # access values in dictionary using keys
        print("ENSMUSG00000063524 locus_tag is: {}".format(gene_dict["ENSMUSG00000063524"]))
```