EEB 177 Lecture 8

Thursday Feb 8th, 2020

–

Preliminaries

- Create a new **jupyter** notebook and save the file "classwork-Thursday-2-6.ipynb" to your class-assignments directory
- push this to your remote repository
- you can write answers to today's exercises in this file.

Topics

- end of collections...
- for, if, else, while statements

Dictionaries

Dictionaries are collections of unordered lists where elements (*values*) are accessed by *keys*. We separate *key-value* pairs using a comma.Dictionaries are defined using {}

```
# create an empty dictionary
my_dict = {}
# strings, floats or even lists can be values in a diction
my_dict = {"a": "test", "b": 3.14, "c": [1, 2, 3,
4]}
GenomeSize = {"Homo sapiens": 3200.0, "Escherichia coli": 4
# A dictionary has no natural order (i.e., the order of ke
GenomeSize
```

```
{'Arabidopsis thaliana': 157.0,
'Escherichia coli': 4.6,
'Homo sapiens': 3200.0}
```

You can access the value of a dictionary by suppling the key for that pair.

```
In [3]: GenomeSize["Arabidopsis thaliana"]
Out[3]: 157.0
```

You can also add entries by supplying a new key-value pair.

```
In [4]: GenomeSize["Saccharomyces cerevisiae"] = 12.1
In [5]: GenomeSize
Out[5]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3200.0,
 "Saccharomyces cerevisiae": 12.1}
```

Adding keys and changing values in dictionaries

```
# ALREADY IN DICTIONARY!
In [6]: GenomeSize["Escherichia coli"] = 4.6
In [7]: GenomeSize
Out[7]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3200.0,
 "Saccharomyces cerevisiae": 12.1}
In [8]: GenomeSize["Homo sapiens"] = 3201.1
In [9]: GenomeSize
Out[9]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3201.1,
 "Saccharomyces cerevisiae": 12.1}
```

copy

```
In [13]: GS = GenomeSize.copy()
In [14]: GS
Out[14]:
{'Arabidopsis thaliana': 157.0, 'Escherichia coli': 4.6,
'Homo sapiens': 3201.1, 'Saccharomyces cerevisiae': 12.1}
```

clear

removes all elements

```
In [15]: GenomeSize.clear()
In [16]: GenomeSize
Out[16]: {}
```

get

gets a value from key

```
In [67]: GenomeSize.get("Homo sapiens")
Out[67]: 3200.0
```

this function is very useful for initializing the dictionary, or to return a special value when the key is not present.

```
In [68]: GenomeSize.get("Mus musculus", 10)
Out[68]: 10
```

items

returns key value pairs. Can be used to print contents of a dictionary.

```
for k,v in GS.items():
    print(k, v)
('Homo sapiens', 3200.0)
('Escherichia coli', 4.6)
('Arabidopsis thaliana', 157.0)
```

keys, values

These functions return lists of the keys or values of the dictionary.

```
In [72]: GS.keys()

Out[72]: ['Homo sapiens', 'Escherichia coli', 'Arabidopsis

In [74]: GS.values()

Out[74]: [3200.0, 4.6, 157.0]
```

Creating dictionaries

You will often create a dictionary and then populate it. Try this!

```
enzymes = {}
enzymes['EcoRI'] = r'GAATTC' # r before the string
#tells python to automatically escape every character
enzymes['AvaII'] =  r'GG(A|T)CC'
enzymes['BisI'] =  r'GC[ATGC]GC'
enzymes.keys()
enzymes.values()
```

You can use **zip()** to turn two lists into a dictionary

```
keys = ('name', 'age', 'food')
values = ('Monty', 42, 'spam')
zip(keys, values) #makes a list of tuples
my_new_dict = dict(zip(keys, values))
my_new_dict
```

sets are lists without duplicate elements

```
In [1]: a = [5,6,7,7,7,8,9,9]
In [2]: b = set(a)
In [3]: b
Out[3]: set([8, 9, 5, 6, 7])
In [4]: c=set([3,4,5,6])
In [5]: b & c
Out[5]: set([5, 6])
In [6]: b | c
Out[6]: set([3, 4, 5, 6, 7, 8, 9])
In [7]: b ^ c
Out[7]: set([3, 4, 7, 8, 9])
```

The operations are: Union | (or); Intersection & (and); symmetric

You can concatenate similar collection elements with +

```
In [1]: a = [1, 2, 3]
In [2]: b = [4, 5]
In [3]: a + b
Out[3]: [1, 2, 3, 4, 5]
In [4]: a = (1, 2)
In [5]: b = (4, 6)
In [6]: a + b
Out[6]: (1, 2, 4, 6)
In [7]: z1 = {1: "AAA", 2: "BBB"}
In [8]: z2 = {3: "CCC", 4: "DDD"}
In [9]: z1 + z2
     ---------------------------------------
     TypeError  Traceback (most recent call last)
     ----> 1 z1 + z2
TypeError: unsupported operand type(s) for +: "dict" and "d
```

collections recap

```
# round brackets --> tuple
In [1]: type((1, 2))
Out[1]: tuple
# square brackets --> list
In [2]: type([1, 2])
Out[2]: list
# curly brackets, sequence of values --> set
In [3]: type({1, 2})
Out[3]: set
# curly brackets, key-value pairs --> dictionary
In [4]: type({1: "a", 2: "b"})
Out[4]: dict
```

for

for loops

imagine you wanted to print out each element of this list

```
apes = ["Homo", "Pan", "Gorilla"]
```

you could print these out seperately

```
print(apes[0] + " is an ape")
print(apes[1] + " is an ape")
print(apes[2] + " is an ape")
```

A better way is to use the **for** loop syntax

```
apes = ["Homo sapiens", "Pan troglodytes", "Gorilla gorilla
for ape in apes:
    print(ape + " is an ape")
```

for loops have the following structure:

```
for x in y:
    do something
```

y is a list (or list-like object) x is a variable names the colon sets off an indented block of code (the body of the loop)

a more complex loop

```
apes = ["Homo", "Pan", "Gorilla"]
for ape in apes:
    name_length = len(ape)
    first_letter = ape[0]
    print(ape + " is an ape. Its name starts with " + first
    print("Its name has " + str(name_length) + " letters")
```

the range function

```
for number in range(6):
    print(number)
```

more on range

With two numbers, range will count up from the 1st number (inclusive) to the second (exclusive):

```
for number in range(3, 8):
    print(number)
```

range and step

With three numbers, range will count up from the 1st to the second with the step size given by the third:

```
for number in range(2, 14, 4):
    print(number)
```

while loops

a while loop runs until some condition is met.

```
count = 0
while count<10:
    print(count)
    count = count + 1
```

program flow in python

In its simplest form, a program is just a series of instructions (statements) that the computer executes one after the other. In Python, each statement occupies one line (i.e., it is terminated by a newline character). Other programming languages use special characters to terminate statements (e.g., ; is used in C).

Lets demonstrate statements that control flow in python with a simple program.

conditional tests

A condition is simply a bit of code that can produce a true or false answer.

```
print(3 == 5)
print(3 > 5)
print(3 <=5)
print(len("ATGC") > 5)
print("GAATTC".count("T") > 1)
print("ATGCTT".startswith("ATG"))
print("ATGCTT".endswith("TTT"))
print("ATGCTT".isupper())
print("ATGCTT".islower())
print("V" in ["V", "W", "L"])
```

we use conditional tests to control the flow of our program

```
expression_level = 125
if expression_level > 100:
    print("gene is highly expressed")
```

if, elif, else

if and **else** create branching points in your program resulting in the execution of different blocks of code depending on a condition.

```
x=4
if x % 2 == 0:
    print("Divisible by 2") #body of loop
```

note indentation to designate loop body

We can use **else** to specify an action when a condition is not met.

```
x=4
if x % 2 == 0:
    print("Divisible by 2")
else:
    print("Not divisible by 2")
```

```
expression_level = 125
if expression_level > 100:
   print("gene is highly expressed")
else:
    print("gene is lowly expressed")
```

If we have multiple cases that need to be evaluated, **elif** is useful...

```
x = 17
if x % 2 == 0:
    print("Divisible by 2")
elif x % 3 == 0:
    print("Divisible by 3")
elif x % 5 == 0:
    print("Divisible by 5")
elif x % 7 == 0:
    print("Divisible by 7")
else:
    print("Not divisible by 2, 3, 5, 7")
```

Pseudocode

A way to plan code without worrying about syntax

Example: Write a program that asks the user for a temperature in Fahrenheit and prints out the same temperature in Celsius.

Pseudocode

```
xx = Get user input


yy = Convert xx to Celsius


Output message displaying Celsius temperature
```

python

```
python
xx = input("What is the temperature in Fahrenheit?")
yy = (float(xx) - 32) * 5 / 9
print("The temperature in Celsius is {}".format(yy)
```

Here is another example showing a loop.

Write a program in which a password is set and the program will keep prompting the user to guess it, until they get the word right.

Pseudocode

```
set password
ask user to try to guess password
    if the password is wrong, say access denied
    if the password is right, grant access
```

python

```
password = ("sesame")
hello = input("Enter the password: ")
while hello != password:
    print ("Access denied")
    hello = input("Enter the password: ")
    if hello == password:
```

Challenge

change the program below so that it reports the temperature in Fahrenheit or Celsius depending on user input.

```
xx = input("What is the temperature in Fahrenheit?")
yy = (float(xx) - 32) * 5 / 9
print("The temperature in Celsius is {}".format(yy)
```

if, elif, and else part II

we can handle complex flow with these statements.

```
file1 = open("one.txt", "w")
file2 = open("two.txt", "w")
accs = ['ab56', 'bh84', 'hv76', 'ay93', 'ap97', 'bd72']
for accession in accs:
    if accession.startswith('a'):
        file1.write(accession + "\n")
    else:
        file2.write(accession + "\n")
```

use **and**, **or** and **not** to specify complex conditionals

```
accs = ['ab56', 'bh84', 'hv76', 'ay93', 'ap97', 'bd72']
for accession in accs:
    if accession.startswith('a') and accession.endswith('3
        print(accession)
```

```
accs = ['ab56', 'bh84', 'hv76', 'ay93', 'ap97', 'bd72']
for accession in accs:
    if accession.startswith('a') or accession.startswith('b
        print(accession)
```

```
accs = ['ab56', 'bh84', 'hv76', 'ay93', 'ap97', 'bd72']
for acc in accs:
    if acc.startswith('a') and not acc.endswith('6'):
        print(acc)
```

```
file1 = open("one.txt", "w")
file2 = open("two.txt", "w")
file3 = open("three.txt", "w")
accs = ['ab56', 'bh84', 'hv76', 'ay93', 'ap97', 'bd72']
for accession in accs:
    if accession.startswith('a'):
        file1.write(accession + "\n")
    elif accession.startswith('b'):
        file2.write(accession + "\n")
    else:
        file3.write(accession + "\n")
```

Reading and writing files

To work with text file contents you need to first open the file and then read the contents in as a string

```
# open the file
my_file = open("dna.txt")
# read the contents
my_dna = my_file.read()
# calculate the length
dna_length = len(my_dna)
# print the output
print("sequence is " + my_dna +  " and length is " + str(dn
```