Basics of programming in python

Key ingredients of programming language:

- data types and data structures
- functions
- control flow (iteration and logical branches)

Key data types in python

- numbers (integers and floating point)
- strings
- lists
- numpy arrays (*)
- dictionaries
- pandas dataframes (*)

Basic examples

From before, remember:

```
n = 56  # integer
m = 1234.48  # floating point
L = [1, 2, 3, 4]  # list
name = "Jeremy"  # string
```

The `typeof` operator tells you what something is.

```
print("type of n is {}, type of name is {}".format(type(n), type(name)))
```

Working with Lists and Strings

Split a string to a list.

```
L = list("My name is Jeremy")
print(L)
```

Join a list to a string.

```
print(''.join(["A","B","C"]))
print('_'.join(["A","B","C"]))
```

Dictionaries

A dictionary (or a HashMap, or an associative array) is like an array with arbitrary subscripts.

```
D = {"first_name": "Jeremy", "last_name": "Teitelbaum"}
D["middle_name"] = "Thau"
print(D["first_name"])
D["Title"] = "Emperor"
```

```
print(D)
# D["Subtitle"]
```

Arrays

```
import numpy as np

x=np.array([1,2,3,4])
x=np.linspace(-5,5,10)
```

Booleans

```
T = True
F = False
print(T or F) # or
print(T and F) # and
3 == 5 # equality
3 > 5 #
3 < 5 #
x = (3 <= 5) #
print(x)
y = (3 != 5) #
print(y)
```

Functions

```
import scipy.stats as sps

def my_function(n,mu,s):
    x = sps.norm.rvs(mu,s,size=n)
    return x
```

Important concepts:

- arguments
- scope
- return values

Scope

Basic rule of scope: Variables created inside functions are completely separate from those outside the function, changing them has no effect.

Exception: some operations (such as list append) modify an element in place and in these cases you may end up modifying something.

```
def f(a,b):
    x=a+b
    return x


x=3
print("before executing f, x={}".format(x))
print(f(2,5))
print("after executing f, x={}".format(x))
```

```
def f(x):
    x=x+["d"]
    return x

L=["a","b","c"]
print("L before is {}".format(L))
print("result of f(L) is {}".format(f(L)))
print("L after is {}".format(L))
```

```
def f(x):
    x.append("d") #
    return x

# Warning
x = ["a","b","c"]
print(f(x))
print(x)
```

```
x = 55

def f(n):
    n = n+x
    return n

f(24)
```

Iteration

```
for x in range(10):
    print(x,end=',')
print('\n---')

for x in ["a","b","c"]:
    print(x,end=',')

# Also available: while
```

Logic

```
if 3<5:
    print("ha")
else:
    print("ba")
```

```
if 3+5==8 and 3-5==-2:
    print("Yeah!")
else:
    print("Nah!")
```

```
if 3+5 in [1,2,3,4,5,6,7]:
    print("Yeah")
else:
    print("nah!")
```

List Comprehensions

This is one of the most useful things about python.

```
L = ["hello","Hello","HELLO","jeremy","jereMy"]
N = [f(x) for x in L]
M = [f(x) for x in L if x[0]=="H"]
print(N,M)
```

Another example.

```
s="Jeremy Teitelbaum"
L=[x for x in list(s) if x not in [" "]]
print(L)
```

Compare:

```
S=""
for x in "Jeremy Teitelbaum":
    if x not in [" "]:
        S+=x
```

A few other tricks

- default arguments
- docstrings

```
def f(x=0,y=1):
    return x+y

print(f())
print(f(1))
print(f(3,4))
```

```
def first_letter_cap(s):
    "Returns s but first letter of string is upper case"
    return s[0].upper()+s[1:]
```

Some examples

Take a string and make its first character upper case and the rest lower.

```
def f(s):
    l = s[0].upper()+s[1:].lower()
    return l

print(f("hello"),f("Hello"),f("HELLO"))
```

Now do this for each element of a list.

```
def h(L):
    N=[]
    for x in L:
        N = N + [f(x)]
    return N

h(["hello","HELLO","jeremy","JEREMY","jerEmy"])
```

Problems

1. Write a function which takes a string and standardizes it by:
    - removes all characters which are not letters, numbers, or spaces
    - makes all the letters lower case
    - replacing all spaces by underscore ’_’

Hint: convert the string to a vector of letters.

1. The `penguins_raw.csv` file can get loaded into a pandas dataframe, which is a fancy type of tabular layout. It has named columns that you can extract with `.columns`

```
import pandas as pd
penguins_raw = pd.read_csv("data/penguins-raw.csv")
penguins_raw.columns
```

You can assign to `penguins_raw.columns` to change the column names. Use your function from part 1 to standardize and simplify the column names.

1. You can access a column of the dataframe using `.`, so for example `penguins_raw.Species` should give you the column species. Replace this column with just the first word of the species name (Gentoo, Adelie, Chinstrap). To do this, you have to use the `map` method. If `f` is a function that picks out the first word of a string, then

`penguins_raw.species.map(f)` returns the result of applying `f` to every element of `penguins_raw.species`.

2. Let n be a positive real number and let x_0 be 1. The iteration

$$x_{k+1} = x_k/2 + n/(2x_k)$$

converges to the square root of n. (This is Newton's Method). Write an R function which computes the square root using this iteration. You should continue to iterate until x_{k+1} is within 10^{-6} of x_k.

```
# def f(n):
#
# ...
```

Suppose you want to save the successive values you computed during the iteration for plotting purposes. How could you do that (and return them)?

Suppose you want the tolerance (here 10^6) to be a parameter?

Suppose you want to set a maximum number of iterations, in case something goes wrong, to prevent an infinite loop?