PHY494 Solution 03

January 24, 2019

(See *assignment_03.pdf* for the detailed break-down of points.)

3.1 Python data types (7 points)

a) Float (float)
b) Integer (int)
c) Boolean (bool)
d) String (str)
e) List (list)
f) Tuple (tuple)
g) NoneType (NoneType)
h) Dictionary (dict)

3.2 Python Lists and Strings (20 points)

(Note: in the following the `ipython` prompts `In[]` and `Out[]` are shown to clearly indicate input and output.)

a) Slicing:

```
In [3]: bag[1:3]
Out[3]: ['towel', 'tea']
```

b) step -1 reverses the list:

```
In [4]: bag[::-1]
Out[4]: [42, 'tea', 'towel', 'guide']
```

To get `['tea', 'towel']`:

```
In [6]: bag[2:0:-1]
Out[6]: ['tea', 'towel']
```

alternatively:

```
In [7]: bag[-2:-4:-1]
Out[7]: ['tea', 'towel']
```

c) String slicing:

```
In [8]: ga[:4]
Out[8]: 'Four'

In [9]: ga[15:20]
Out[9]: 'seven'
```

d) list assignments

i) `bag[0] = book` sets the first element of the list to the string "book".

ii) Other lists:

```
In [14]: bag
Out[14]: ['book', 'towel', 'tea', 'mice']

In [15]: mybag
Out[15]: ['book', 'towel', 'tea', 'mice']

In [16]: yourbag
Out[16]: ['book', 'towel', 'tea', 42, 'money']
```

iii) `x = a` makes `x` identical to `a`. Any change in `x` is reflected in `a`. (In C one would say, `x` points to the same address as `a`.)
`y = a[:]` makes a (shallow) copy of `a` and assigns it to `y` that is now independent from `a`.

e) String manipulation

i) assignment
`ga[:4] = "Three"`
raises a `TypeError`: one cannot assign to parts of a string. This is how strings differ from lists.

ii) string manipulation
Create a new string

```
In [21]: 'Three' + ga[4:]
Out[21]: 'Three score and seven years ago'
```

or use the string's replace() method:

```
ga.replace("Four", "Three")
```

f) String methods

Splits the string on whitespace:

```
In [3]: ga.split()
Out[3]: ['Four', 'score', 'and', 'seven', 'years', 'ago']
```

Assign slice of the splitted list to individual variables (using tuple assignment):

```
In [4]: a, b, c = ga.split()[:3]
In [5]: print(a,b,c)
('Four', 'score', 'and')
```

`list()` turns a list into a list (d'oh!):

```
In [6]: list([1,2,3])
Out[6]: [1, 2, 3]
```

`list()` turns a string into a list of characters (aha!):

```
In [7]: list(ga)
Out[7]:
['F',
 'o',
 'u',
 'r',
 ' ',
 's',
 'c',
 'o',
 'r',
 'e',
 ' ',
 'a',
 'n',
 'd',
 ' ',
 's',
 'e',
 'v',
 'e',
```

```
'n',
' ',
'y',
'e',
'a',
'r',
's',
' ',
'a',
'g',
'o']
```

(The fact that the list is printed vertically instead of horizontally `['F', 'o', 'u', ...]` is not relevant and just depends on how `python` or `ipython` are configured to print variables to the screen.)

In summary: Strings can be processed by `split()` (splits on white space by default) and the resulting list assigned to individual variables. The optional argument to `split()` can designate a different character to split on. Note that the default `split(None)` splits on *any* number of consecutive white space characters whereas `split(" ")` splits on *each* white space character so that :

```
"  1 2 ".split() ==  ["1", "2"]
"  1 2 ".split(" ") == ["", "", "1", "2", ""]
```

Strings can be turned into a list of characters by using the list constructor `list()`.

g) Nested list:

```
In [23]: bags[0]
Out[23]: ['salt', 'pepper']

In [24]: bags[0][1]
Out[24]: 'pepper'

In [25]: bags[1][2]
Out[25]: 'ruler'
```

3.3 Loops (14 points)

a)
```
sentence = ["We", "must", "walk", "before", "we", "can", "run"]

for i in sentence:
    print(i)
```

b) BONUS: use the "step" argument for slicing a list `sentence[start:stop:step]`:

```
sentence = ["We", "must", "walk", "before", "we", "can", "run"]

for i in sentence[::2]:
    print(i)
```

c) Solution 1: use the start and stop argument of `range(start, stop)` and remember that stop is *excluded* (so we need to use 1001 for stop):

```
total = 0
for i in range(1, 1001):
    total += i

print(total)
```

Solution 2: Add 1 in the loop (but this is less clear than the Solution 1 above)

```
result = 0
for i in range(1000):
    result += i+1

print(result)
```

Solution 3: (hardcore solution - learn about "Python list comprehensions"):

```
total = sum([i for i in range(1, 1001)])
print(total)
```

(Look up "Python list comprehension"!)

d) Countdown with loops

i) Canonical countdown

```
counter = 10
while counter > 0:
      print(counter)
      counter -= 1
```

ii) Multiple solutions. First use all arguments of `range()`:

```
for i in range(10, 0, -1):
    print(i)
```

Using arithmetic:

```
for i in range(10):
    print(10-i)
```

3.4 Simple coordinate manipulation in Python (11 points)

Complete solutions are provided in the files `Solution/coordinates_{a,b,c,d}.py`.

a) see `Solution/coordinates_a.py`:

```
[1.34234, 1.34234, 0.0]
```

b) see `Solution/coordinates_b.py`

```
1.34234
```

c) see `Solution/coordinates_c.py`; however, many different solutions are possible and some are listed here:

Here we make a copy of the positions and then increment each element *in place* with `new_position[i] += t[i]`:

```
new_positions = []
for position in positions:
    new_position = position[:]  # make a copy
    for i in range(3):
        new_position[i] += t[i]
    new_positions.append(new_position)
print(new_positions)
```

If you have something like the above or the solution in `Solution/coordinates_c.py` then you get all points.

However, that is pretty ugly (and un-pythonic) code, and it has poor performance. The following are all better ways to do the same thing: try to understand them!

This can be written more efficiently with list comprehensions

```
new_positions = [[position[i] + t[i] for i in range(3)]
                 for position in positions]
```

Use of the `zip()` function can also help:

```
new_positions = [[xi + ti for xi, ti in zip(position, t)]
                 for position in positions]
```

(We will see in the lesson on NumPy that nothing beats `numpy` in terms of clarity and speed, though.)