Lab 5: While

Feb 8, 2018

Main Event

1. if-reduction

In a single Python file, make two copies of the following code:

```
x = 3
if x < 2:
  print('a')
else:
  if x < 4:
    print('b')
  else:
    if x < 6:
      print('c')
    else:
      print('d')
print(x)
```

Leave the first copy as is.

In the second, do two things:

1. Remove `x=3`. You can rely on the definition of `x` from the first copy.
2. Simplify the code using `elif` statements. Your solution should remove the nested blocks entirely.

Convince yourself that your solution is correct by changing value of `x` and ensuring that the same thing is printed twice.

Solution:

```
x = 3

# original
if x < 2:
    print('a')
else:
    if x < 4:
        print('b')
    else:
        if x < 6:
            print('c')
        else:
            print('d')
print(x)

# consolidated
if x < 2:
    print('a')
elif x < 4:
    print('b')
elif x < 6:
    print('c')
else:
    print('d')
print(x)
```

2. FizzBuzz

The “FizzBuzz” programming challenge is as follows:

> *Write a program that prints the integers from 1 to 100, but for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers that are multiples of both three and five print “FizzBuzz”.*

The problem is well known within the programming community; so much so that, during a technical interview, it is a common question to ask a candidate. As such, there are a plethora—a cornucopia even—of available solutions online (one site displays six different ways to do this in Python!). It is *strongly recommended* you spend some time on this without referring to the internet: it’s much better to get it wrong here, but understand it, than to shortcut and later lose out on a job.

Solution:

```
fizz = 'Fizz'
buzz = 'Buzz'

for i in range(1, 100 + 1):
    if i % 15 == 0: # must come first or it's never run!
        fb = fizz + buzz
    elif i % 3 == 0:
        fb = fizz
    elif i % 5 == 0:
        fb = buzz
    else:
        fb = str(i)
    print(fb)
```

3. List counting

Consider the following list-of-lists:

```
x = [[1, 2], [3], [4, 5, 6, 7], [8, 9]]
```

Using a for-loop:

1. Count the number of even values in the list.
2. Count the number of odd values contained in sublists with more than two elements.

Solution:

```
x = [[1, 2], [3], [4, 5, 6, 7], [8, 9]]

# even values
evens = 0
for sublist in x:
    for i in sublist:
        if i % 2 == 0:
            evens += 1
print('Even values:', evens)

# odd values where cardinality is greater-than 2
odds = 0
```

```
for sublist in x:
    if len(sublist) > 2:
        for i in sublist:
            if i % 2 == 1:
                odds += 1
print('Odd values (more than two elements):', odds)
```

4. Loop addition

Write a while-loop that adds the numbers 0 through 5 (inclusive).

Solution:

```
i = 0
total = 0
while i <= 5:
    total += i
    i += 1
print(total)
```

5. (More) String reversal

Consider the following string:

```
x = 'Billie Jean is not my lover'
```

Reverse the string using

1. a for-loop, then
2. a while-loop.

You cannot use slicing or string methods!

Solution:

```
initial = 'Billie Jean is not my lover'

# 1. Using a for-loop (from the previous lab!)
reverse = ''
for i in initial:
    reverse = i + reverse
print(reverse)
```

```
# 2. Using a while-loop. Because the while-loop forces the programmer
#    to manually control loop iteration, there are numerous ways to do
#    this!

# Approach 1: model the while-loop after the for-loop
reverse = ''
i = 0
while i < len(initial):
    reverse = initial[i] + reverse
    i += 1
print(reverse)

# Approach 2: use backwards indexing
reverse = ''
i = len(initial)
while i > 0:
    reverse += initial[i - 1]
    i -= 1
print(reverse)
```

6. Friend book

Repeatedly ask the user for the names of their friends. When the user specifies a particular value, stop asking and display all of the names that they have entered.

```
Enter your friends name, or 'stop' to finish: Bob
Enter your friends name, or 'stop' to finish: Paul
Enter your friends name, or 'stop' to finish: Caleb
Enter your friends name, or 'stop' to finish: stop
You have 3 friends. They are:
Bob,
Paul, and
Caleb
```

Notice that the program has listed the number of friends a person has, along with a comma after each name, and the word “and” to signify the last name is next—you should do the same.

Solution:

```
friends = []
while True:
    name = input("Enter your friends name, or 'stop' to finish: ")
    if name == 'stop':
        break
    friends.append(name)

posse = len(friends)
print('You have ', posse, 'friends. They are:')

i = 0
while i < posse:
    if i == posse - 1: # we're at the last element!
        suffix = ''
    else:
        suffix = ','

    if i == posse - 2: # we're at the second to last element
        suffix += ' and'

    print(friends[i], suffix, sep='')
    i += 1
```

Additional Practice

1. List counting remix

Repeat the list counting exercise, but

1. use a while-loop instead of a for-loop;
2. create your own list: start with an empty list, then use a for-loop to populate that list with lists of integers. The inner lists should be of random length, and can contain random (integer) values.

Solution:

```
import random

# 1: Using a while loop
```

```
x = [[1, 2], [3], [4, 5, 6, 7], [8, 9]]

# even values
evens = 0
i = 0
while i < len(x):
    sublist = x[i]
    j = 0
    while j < len(sublist):
        if x[i][j] % 2 == 0:
            evens += 1
        j += 1
    i += 1
print('Even values:', evens)

# odd values where cardinality is greater-than 2
odds = 0
i = 0
while i < len(x):
    length = len(x[i])
    if length > 2:
        j = 0
        while j < length:
            if x[i][j] % 2 == 1:
                odds += 1
            j += 1
    i += 1
print(odds)

# 2: Using the randomly generated list
x = []
for i in range(random.randrange(1, 10)):
    y = []
    for j in range(random.randrange(1, 5)):
        y.append(random.randrange(20))
    x.append(y)

    # or, simply,
    # x.append(random.sample(range(20), random.randrange(1, 5)))

# ... counting the even and odds remains the same
```

2. Create your own replace

This exercise will take you through a series of steps to recreate the string replace method. Our version of replace will only operate on alphabetic characters. You should submit something unique (a separate file is best) for each sub-question.

Single character, single casing

Start with the static case to get your *algorithm* together. Build your solution using the following skeleton code:

```
source = 'Adam made him take a placard'
s1 = 'a'
s2 = 'z'
# insert code here to create target
print(target) # should produce 'Adzm mzde him tzke z plzczrd'
```

Note that here we're doing a literal replace: "a" is not the same character as "A".

Single character, multiple casing

Update your code such that the replacement character replaces all instances of a given character, regardless of its casing; however, in the output that replacement will respect the casing. Continuing out example above, note the difference in `print(target)`:

```
source = 'Adam made him take a placard'
s1 = 'a'
s2 = 'z'
# insert code here to create target
print(target) # should produce 'Zdzm mzde him tzke z plzczrd'
```

Here "A" was replaced with "Z", the upper case version of `s2`.

User input

Update the code such that `source`, `s1`, and `s2` are specified by the user. An example interaction:

```
Your string: Adam made him take a placard
Unwanted character: a
Replacement character: z
```

```
Zdzm mzde him tzke z plzczrd
```

Multiple character, multiple casing, user input

Update the code such that we replace a series of characters. In this case, `s1` is no longer assumed to be a single character: the user will specify a string; for each character *c* in that string, we replace instances of *c* in `source` with `s2`. An example interaction:

```
Your string: Adam made take a placard
Unwanted character(s): ae
Replacement character: z
Zdzm mzdz tzkz z plzczrd
```

Solution:

```
# single character, single casing
source = 'Adam made him take a placard'
s1 = 'a'
s2 = 'z'

target = ''
for i in source:
    if i == s1:
        target += s2
    else:
        target += i
print(target)

# single character, multiple casing
target = ''
for i in source:
    if i == s1:
        target += s2
    elif i == s1.upper():
        target += s2.upper()
    else:
        target += i
print(target)

# user input
source = input('Your string: ')
```

```
s1 = input('Unwanted character: ').lower()
s2 = input('Replacement character: ').lower()

target = ''
for i in source:
    if i == s1:
        target += s2
    elif i == s1.upper():
        target += s2.upper()
    else:
        target += i
print(target)

# multiple character, multiple casing, user input
source = input('Your string: ')
s1_sequence = input('Unwanted character(s): ').lower()
s2 = input('Replacement character: ').lower()

for s1 in s1_sequence:
    target = ''
    for i in source:
        if i == s1:
            target += s2
        elif i == s1.upper():
            target += s2.upper()
        else:
            target += i
    source = target # this is the key!
print(target)
```

Introduction to Computer Science

Introduction to Computer Science
jerome.white@nyu.edu

jerome-white

Learning computer science concepts through practice.