Name: Date:

Lesson 4: Recap and Robot Interlude

strings, slicing, functions

In Lesson 1 we introduced strings, string slicing and basic functions. Let’s recap what we learnt.

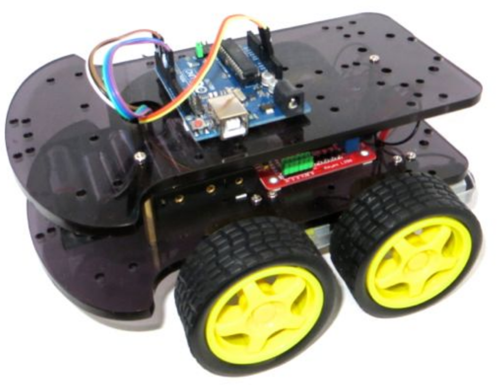
TRY IT OUT #1: Write a function `double()` that takes either a string or an integer and returns the double of that input. So for instance `double(3)` would return 6 and `double(“hi”)` would return “`hihi`”

lists, for loops, control flow

In Lesson 2 we introduced lists. We also introduced some key program control structures: for loops, if statements, <, >, ==. We can use these control elements together with strings and lists to

TRY IT OUT #2: Write a function called `doubleList()` that takes a list of integers and returns it with each element of the list doubled. Call this function with `range(10)`

And now for something completely different! Meet Orion the robot! He was built from a kit using a guide available here: http://orionrobots.co.uk/construction_guide.html.



The brain inside Orion is an Arduino microcontroller which can be programmed using a special language called the Arduino Programming language via USB from your PC. A program has been written and transferred to Orion which allows you to control its movement in four directions: front, back, left, right. The Arduino is wired up to a motor controller board which is connected to motors that drive the four wheels. There is a battery pack with 6 AA batteries supplying power and a Bluetooth dongle connected to the microcontroller. There are two ways to talk to Orion – via USB or Bluetooth. The Python code below can be used to communicate between a PC and handset via USB. One new Python built-in function we are using here is `raw_input()` which prints a string and then waits and returns user input on the keyboard. The Android app being used to control the robot is written in Java but operates the same way – you send WASD characters to Orion to make him move in different directions.

```
port=int(raw_input('Enter COM port:'))
reading=True
ser=serial.Serial(port='COM%d' % port)   # COM port
while reading:
    key=raw_input('>')
    c=key.lower()
    if c=='w':
        ser.write('w')
    elif c=='a':
        ser.write('a')
    elif c=='s':
        ser.write('s')
    elif c=='d':
        ser.write('d')
    elif c in ['x',' ']:
        print("Bye...")
        reading=False
ser.close()
```

TRY IT OUT #5: Let's play with Orion using the Android phone app. Notice how the key settings are encoded in the app. Can you see how it is working?

ord, chr, bool

In Lesson 3 we introduced `ord()` and `chr()` as well as bool values `True` and `False`. These are the additional elements that we need to help us implement the Caesar Cipher.

TRY IT OUT #4: Try working through the Caesar Cipher Code Quest from last week. You will need to understand and implement a function called `encodeCharacter()` that breaks in input string into individual characters and then calls `shiftCharacter()` as on each of them.

```
def shiftCharacter(c,offset):
    if c.isalpha():
        n=ord(c)
        n=n+offset
        if n > ord('z'):
            n=n-26
        elif n < ord('a'):
            n=n+26
        translated=chr(n)
    else:
        translated=chr(c)
    return translated

s=''
offset=2
for c in 'abcdefghijklmnopqrstuvwxyz':
    s=s+shiftCharacter(c,offset)
print(s)
```

Bringing it all together ….

We can extend the Caesar cipher by using a **key** for our next Secret Code Quest. This is the basis of the **Vigenere cipher** that was widely used for 300 years until 150 years ago: http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher . The Vigenere cipher was called "**le chiffre indéchiffrable**" by the French – the "indecipherable cipher".

SECRET CODE QUEST!

Create a key (eg. '1234') which you can use to set a variable offset value for each call to `shiftCharacter (s,offset)`. You will need to break the key into individual integer offsets to pass in. You can use the `list()` built in function to help you do this:

```
list('1234')
['1','2','3','4']
for x in list('1234'):
    int(x)
```

Once you have this working, you could write out your encoded messages on paper and use Python to help you “decode” them back to their unreversed form. To do that you will need to call `shiftCharacter ()` with the encoded string plus the **negative** of the original offset.

```
def double(s):
    return s*2

def doubleList(ls):
    r=[]
    for el in ls:
        r.append(double(el))
    return r

def decodeCaesar(s,offset):
    r=''
    for c in s:
        r+=shiftCharacter(c,-offset)
    return r

def encodeCaesar(s,offset):
    r=''
    for c in s:
        r+=shiftCharacter(c,offset)
    return r

def shiftCharacter(c,offset):
    if c.isalpha():
        n=ord(c)
        n=n+offset
        if n > ord('z'):
            n=n-26
        elif n < ord('a'):
            n=n+26
        translated=chr(n)
    else:
        translated=c
    return translated

print(double(2))
print(double("hi"))
print(doubleList(range(10)))
offset=3
r=encodeCaesar('this is a secret message!',offset)
print(r)
s=decodeCaesar(r,offset)
print(s)

----------- OUTPUT -------------
4
hihi
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
wklv lv d vhfuhw phvvdjh!
this is a secret message!
```