MY470 Computer Programming

For Loops and List Comprehensions in Python

Week 3 Lab, MT 2017

Control Flow and Indentation

```
for i in list:
    # inside the for loop
    if something is TRUE:
        # inside the if statement
        do something
    else:
        # inside the else statement
        do something
    # inside the for loop, but not in the if or else statement anymore
    do something
# outside of the for loop
do something
```

In [1]:
```
# Exercise: Create a list that contains all integers from 1 to 100 (inclusive), ex
cept that it has the string 'boo'
# for every integer that is divisible by 3
# Your list should look like: [1, 2, 'boo', 4, 5, 'boo', 7, 8, 'boo', 10, ...]
```

In [2]:
```
# Exercise: Sum the even integers from the list below.
lst = [1, 3, 2, 4.5, 7, 8, 10, 3, 5, 4, 7, 3.33]
```

List Comprehensions

Create a new list based on another one.

With a for loop:

```
newlst = []
for i in lst:
    if something is TRUE:
        j = do something to i
        newlst.append(j)
```

With a list comprehension:

```
newlst =  [do something to i for i in lst if something is TRUE]
```

In [3]:
```
# Exercise: Using a list comprehension, create a new list containing
# the squares of the integers in the list below
lst = [1, 3, 2, 4.5, 7, 8, 10, 3, 5, 4, 7, 3.33]
```

In [4]:
```
# Exercise: Consider the lists x and y below. Using a list comprehension,
# create a list that contains all combinations of (elem_x, elem_y)
# such that elem_x + elem_y = 6
# Your answer should look as follows: [(0, 6), (1, 5), (2, 4), (3, 3)]
x = [0, 1, 2, 3]
y = [3, 4, 5, 6]
```

In [5]:
```
# Exercise: Using nested list comprehensions and range(), create a list
# that looks as follows: [[0, 1, 2, 3], [1, 2, 3], [2, 3], [3]]
```

Iterating over Dictionaries

In [12]:
```
letters = {'a':'apple', 'b': 'beetle', 'c': 'cat'}

for i in letters:  # equivalent to: for i in letters.keys():
    print(i)

print()
for i in letters:
    print(letters[i])
# equivalent to:
for i in letters.values():
    print(i)

print()
for i in letters.items():
    print(i)

print()
for i,j in letters.items():
    print(i, ":", j)
```

```
a
b
c

apple
beetle
cat
apple
beetle
cat

('a', 'apple')
('b', 'beetle')
('c', 'cat')

a : apple
```

In [7]:
```
# Exercise: Using a dictionary comprehension, create a new dictionary that
# contains the keys from dictionary letters that are strings, with the value for
# each key assigned to be an empty list
# The new dictionary should look as follows: {'a':[], 'b':[], 'c':[], 'd':[]}

letters = {'a':'apple', 4: None, 'b': 'beetle', 'c': 'cat', 2: None, 'd': 'diamon
d'}
```

In [8]:
```
# Exercise: Now, distribute the words from the list below to the new dictionary
# according to their first letter

dic = {'a':[], 'b':[], 'c':[], 'd':[]}
wordlst = ['a', 'be', 'an', 'the', 'can', 'do', 'did', 'to', 'been']
```

Beware of Iterating over Unordered Collections

- (In Python 3.6 dictionaries are now implemented as ordered but you should not rely on this!)
- For unordered collections, the ordering of elements is determined by how the elements are stored in memory

In [11]:
```
lst = [1, 2, 4, 8, 1, 2]
st = set(lst)

print("List:", [i for i in lst])
print("Set:", [i for i in st])
```

```
List: [1, 2, 4, 8, 1, 2]
Set: [8, 1, 2, 4]
```

Avoid Mutating a List When Iterating over It

In [10]:
```
lst = [i for i in range(10)]
for i in lst:
    popped = lst.pop(i)
    print(popped, lst)
```

```
0 [1, 2, 3, 4, 5, 6, 7, 8, 9]
3 [1, 2, 4, 5, 6, 7, 8, 9]
6 [1, 2, 4, 5, 7, 8, 9]
8 [1, 2, 4, 5, 7, 9]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-10-368f10c34dd9> in <module>()
      1 lst = [i for i in range(10)]
      2 for i in lst:
----> 3     popped = lst.pop(i)
      4     print(popped, lst)

IndexError: pop index out of range
```