**Kernel:** Python 3 (system-wide)

# List comprehensions

A list comprehension is a way to create a new Python `list` from any *iterable* structure (often another list, but not necessarily). The syntax looks a lot like the construction of a list with known components.

```
[expression for variable in iterable if boolean_expression]
```

where `if boolean` is optional.

Here are a couple of examples.

First, we create a `list` from a `range`

In [1]:
```
l = [i for i in range(10)]
print(type(l))
print(l)
```

Out[1]:
```
<class 'list'>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Why would we want to do this? Python **ranges** are different than **lists**; in particular, while they are *iterable*, they are not *mutable*.

In [4]:
```
r = range(10)
print(r)
print(type(r))
r.append(10)
```

Out[4]:
```
range(0, 10)
<class 'range'>
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_759/2580088131.py in <module>
      2 print(r)
      3 print(type(r))
----> 4 r.append(10)

AttributeError: 'range' object has no attribute 'append'
```

Next, let's compute the squares of the first 10 positive integers.

In [5]:
```
[x**2 for x in range(10)]
```

Out[5]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Now, let's use a list comprehension to write a *function* that takes any *iterable* object as its first argument, and returns a `list` of the values in that first argument that are **divisible** by the second argument.

In [6]:
```
def divisible_by (iterable, divisor):
    return [a for a in iterable if a % divisor == 0]
```

And now let's see if it works!

In [8]:
```
r = range(21)

evens = divisible_by(r, 2)
print(evens)

by_seven = divisible_by(r, 7)
print(by_seven)
```

Out[8]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[0, 7, 14]
```

It's not just numbers that we can process using list comprehensions. The following set of examples come from this site. First we define a string to operate on.

In [9]:
```
string = "Practice Problems to Drill List Comprehension in your Head"
```

Now let's write a list comprehension that finds the length of each of the words

In [11]:
```
[len(w) for w in string.split(" ")]
```

Out[11]:
```
[8, 8, 2, 5, 4, 13, 2, 4, 4]
```

Figure out the longest word in the sentence

In [13]:
```
words = string.split(" ")
word_lengths = [len(w) for w in words]
longest = max(word_lengths)
print(longest)
[w for w in words if len(w) == longest]
```

Out[13]:
```
13

['Comprehension']
```

In [14]:
```
[w for w in words if len(w) == max([len(w) for w in words])]
```

Out[14]:
```
['Comprehension']
```

How many **spaces** are there in `string`?

In [16]:
```
len([c for c in string if c == ' '])
```

Out[16]:
```
8
```

It's impressive what we can read without **vowels**. What does `string` look like without the vowels?

In [18]:
```
''.join([c for c in string if c not in ['a', 'e', 'i', 'o', 'u']])
```

Out[18]:
```
'Prctc Prblms t Drll Lst Cmprhnsn n yr Hd'
```

Find all of the words in `string` that are less than 5 letters

In [19]:
```
[w for w in string.split(' ') if len(w) < 5]
```

Out[19]:
```
['to', 'List', 'in', 'your', 'Head']
```

Write this as a **function**, and call it to get the same result as above.

In [20]:
```
def words_less_than(string, delimiter=' ', length=5):
    return [w for w in string.split(delimiter) if len(w) < length]
```

In [22]:
```
print(words_less_than(string, length=5))
print(words_less_than(string, length=9))
```

Out[22]:
```
['to', 'List', 'in', 'your', 'Head']
['Practice', 'Problems', 'to', 'Drill', 'List', 'in', 'your', 'Head']
```

Suppose there was no list comprehension in Python. How could we write it as a **function** ourselves?

In [26]:
```
def list_comprehension(iterable, expression_function=None, contains_function=None ):
    result = []
    for i in iterable:
        if contains_function == None or contains_function(i)==True:
            if expression_function == None:
                result.append(i)
            else:
                result.append(expression_function(i))
    return result
```

Try it out a few times.

In [29]:
```
print(list_comprehension(range(0,10)))
print(list_comprehension(range(0,10), expression_function=lambda x: x**2))
print(list_comprehension(range(0,10), expression_function=lambda x: x**2,
contains_function=lambda x: x < 5))
```

Out[29]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16]
```

Finally, what if we give some default arguments for `expression_function` and `contains_function`.

In [30]:
```
def list_comprehension(iterable, expression_function=lambda x:x,
contains_function=lambda x:True):
    result = []
    for i in iterable:
        if contains_function(i)==True:
            result.append(expression_function(i))
    return result
```

Try it out a few times.

In [31]:
```
print(list_comprehension(range(0,10)))
print(list_comprehension(range(0,10), expression_function=lambda x: x**2))
print(list_comprehension(range(0,10), expression_function=lambda x: x**2,
contains_function=lambda x: x < 5))
```

Out[31]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16]
```

In [0]:
```
```