MY470 Computer Programming

Working with Strings and Lists in Python

Week 2 Lab

Variables

Variables associate objects (values) with a name. Objects have types (belong to classes). Here are the rules for naming variables:

- Variables must begin with a letter (a - z, A - Z) or underscore (_)
- Variables can contain letters, underscore, and numbers

- Watch out for reserved words and names of functions!

In [4]:
```
# List of reserved words in Python: and, as, assert, break,
# class, continue, def, del, elif, else, except, exec,
# finally, for, from, global, if, import, in, is, lambda, not,
# or, pass, print, raise, return, try, while, with, yield

trial = 2
try = 3

# Note that the error below explains what the error is, and where exactly it is located.
```

```
  File "<ipython-input-4-3ecf8fb15fb9>", line 7
    try = 3
        ^
SyntaxError: invalid syntax
```

In [23]:
```
list = [1, 2, 3] # Note the color of "list" - Python recognizes this but you are redefining it!
list((10, 20, 30)) # The in-built function will no longer work
```

```
[1, 2, 3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-23-e0cbabb1e82c> in <module>
      1 #list = [1, 2, 3] # Note the color of "list" - Python recognizes this but you are redefining it!
      2 print(list)
----> 3 list((10, 20, 30)) # The in-built function will no longer work

TypeError: 'list' object is not callable
```

Style Guide for Python Code (PEP 8)

- ☐ Use `UPPERCASE_WITH_UNDERSCORES` for constants, like passwords or secret keys
- ☐ Use `lowercase_with_underscore` for variable names, functions, and methods
- ☐ Use `UpperCamelCase` for classes (coming in Week 5!)

Resources

In addition to the Python resources online, you can query any object to get help on what methods are available

In [3]:
```
dir(dict)
help(dict.popitem)
```

```
Help on method_descriptor:

popitem(self, /)
    Remove and return a (key, value) pair as a 2-tuple.

    Pairs are returned in LIFO (last-in, first-out) order.
    Raises KeyError if the dict is empty.
```

Strings

- Ordered sequences of characters
- Immutable

In [5]:
```
x = 'my string'

# Capitalises the first character (chr) of the string
x = x.capitalize()

# prints the string
print(x)

# prints the chr at index 3
print(x[3])

# prints the last chr
print(x[-1])

# print a range
# NOTE: not inclusive of the last index, 4 chrs because python starts at 0
print(x[0:4])

# Index one to the last index
```

```
# Again, not inclusive
print(x[1:-1])

# EXTENDED SLICING
# Get every other (2) item in the string.
print(x[::2])

# Reverse steps, every other chr
print(x[::-2])
```

```
My string
s
g
My s
y strin
M tig
git M
```

In [2]:
```
# Exercise 1: Make three new strings from the first and last,
# second and second to last, and third and third to last letters
# in the string below. Print the three strings.

p = 'redder'
```

In [3]:
```
# Exercise 2: Make a new string that is the same as string1 but
# with the 8th and 22nd characters missing.

string1 = 'I cancelled my travelling plans.'
```

String Methods

- `S.upper()`
- `S.lower()`
- `S.capitalize()`
- `S.find(S1)`
- `S.replace(S1, S2)`
- `S.strip(S1)`
- `S.split(S1)`
- `S.join(L)`

Methods Can Be "Stringed"

`sls = s.strip().replace('  ', ' ').upper().split()`

However, be aware that this may reduce the clarity of your code.

📖 It is largely a question of code legibility.

⚡ Except when you are working with large data — it is then also a question of memory.

In [4]:
```
# Exercise 3: Remove the trailing white space in the string below,
# replace all double spaces with single space, and format to a sentence
# with proper punctuation. Print the resulting string.

string1 = '  this  is a very badly.  formatted string -  I would  like to make it cleaner\n'
```

In [6]:
```
# Exercise 4: Convert the string below to a list

s = "['apple', 'orange', 'pear', 'cherry']"
```

In [7]:
```
# Exercise 5: Reverse the strings below.

s1 = 'stressed'
s2 = 'drawer'
```

Lists

- Ordered sequence of values
- Mutable

In [3]:
```
mylist = [1, 2, 3, 4]
mylist.append(5)
print(mylist)
```

```
[1, 2, 3, 4, 5]
```

List Methods

- `L.append(e)`
- `L.extend(L1)`
- `L.insert(i, e)`
- `L.remove(e)`
- `L.pop(i)`
- `L.sort()`
- `L.reverse()`

In [32]:
```
# Exercise 6: Use a list operation to create a list of ten elements,
# each of which is '*'
```

In [2]:
```
# Exercise 7: Assign each of the three elements in the list below
```

```
# to three variables a, b, c
ls = [['dogs', 'cows', 'rabbits', 'cats'], 'eat', {'meat', 'grass'}]
```

In [1]:
```
# Exercise 8: Replace the last element in ls1 with ls2
ls1 = [0, 0, 0, 1]
ls2 = [1, 2, 3]
```

In [12]:
```
# Exercise 9: Create a new list that contains only unique elements from list x

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

In [21]:
```
# Exercise 10: Print the elements that occur both in list a and list b

a = ['red', 'orange', 'brown', 'blue', 'purple', 'green']
b = ['blue', 'cyan', 'green', 'pink', 'red', 'yellow']
```

In [14]:
```
# Exercise 11: Print the second smallest and the second largest numbers
# in this list of unique numbers

x = [2, 5, 0.7, 0.2, 0.1, 6, 7, 3, 1, 0, 0.3]
```

In [15]:
```
# Exercise 12: Create a new list c that contains the elements of
# list a and b. Watch out for aliasing - you need to avoid it here.

a = [1, 2, 3, 4, 5]
b = ['a', 'b', 'c', 'd']
```

Week 2 Assignment (SUMMATIVE)

- Practice string and list manipulations
- Practice working with data