Lesson2

October 2, 2019

1 Python Lists

A list is exactly what it sounds like. Lists can contain any kind of objects, as long as they're between square brackets. That's right—a list is both an object AND a container of other objects. This might be confusing at first, but you'll come to learn that it's very convenient to be able to treat one item in a list similarly to how you might treat the list itself.

In lists:

- the order stays the same
- you can get an item by referring to its position in the list, a number called the index

[14]:
```
my_list = ['New York', 'London', 'Budapest']
```

[15]:
```
type(my_list)
```

[15]:
```
list
```

[5]:
```
print(my_list)
```

```
['New York', 'London', 'Budapest']
```

[6]:
```
my_list.append('Bombay')
```

[7]:
```
print(my_list)
```

```
['New York', 'London', 'Budapest', 'Bombay']
```

[8]:
```
my_list[3]
```

[8]:
```
'Bombay'
```

[9]:
```
my_list[3] = 'Mumbay'
```

[10]:
```
print(my_list)
```

```
['New York', 'London', 'Budapest', 'Mumbay']
```

[12]:
```
print(my_list*2)
```

```
['New York', 'London', 'Budapest', 'Mumbay', 'New York', 'London', 'Budapest',
'Mumbay']
```

[16]:
```
print(my_list+["Glasgow"])
```

```
['New York', 'London', 'Budapest', 'Glasgow']
```

[19]:
```
print(my_list)
```

```
['New York', 'London', 'Budapest']
```

[20]:
```
my_list = my_list + ['Glasgow']
```

[21]:
```
for elem in my_list:
    print(elem)
```

```
New York
London
Budapest
Glasgow
```

[26]:
```
for elem in my_list:
    if len(elem)>7:
        print(elem)
```

```
New York
Budapest
```

2 Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

[27]:
```
continents = ('America', 'Europe', 'Asia')
```

[28]:
```
print(continents)
```

```
('America', 'Europe', 'Asia')
```

[29]:
```
continents = continents + ('Africa')
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-29-8761aaedc120> in <module>
----> 1 continents = continents + ('Africa')


TypeError: can only concatenate tuple (not "str") to tuple
```

[31]:
```
continents[2] = 'Africa'
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-31-1c6529c86c21> in <module>
----> 1 continents[2] = 'Africa'


TypeError: 'tuple' object does not support item assignment
```

3 Dictionaries

Dictionaries are also what they sound like - a list of definitions that correspond to unique terms.

Dictionaries are unordered - Dictionary values are accessed by keys - Keys and values are to a dictionary what words and their definitions are to an English dictionary. Each entry in a dictionary is called a key-value pair. To create a dictionary, use curly braces:

[33]:
```
stock_prices = {
    'Tesla' : 240,
    'Apple' : 220,
    'Microsoft' : 130
}
```

[34]:
```
print(stock_prices)
```

```
{'Tesla': 240, 'Apple': 220, 'Microsoft': 130}
```

[37]:
```
stock_prices['Tesla']
```

[37]:
```
240
```

[46]:
```
for key in stock_prices.keys():
    print('The price of ' + key + ': ' + str(stock_prices[key]))
```

```
The price of Tesla: 240
The price of Apple: 220
The price of Microsoft: 130
```

[45]:
```
for key,val in stock_prices.items():
    print('The price of ' + key + ': ' + str(val))
```

```
The price of Tesla: 240
The price of Apple: 220
The price of Microsoft: 130
```

[47]:
```
stock_prices['Microsoft'] = 135
```

[48]:
```
print(stock_prices)
```

```
{'Tesla': 240, 'Apple': 220, 'Microsoft': 135}
```

[59]:
```
stock_prices['Oil'] = 50
```

[50]:
```
print(stock_prices)
```

```
{'Tesla': 240, 'Apple': 220, 'Microsoft': 135, 'Oil': 50}
```

[60]:
```
stock_prices.pop('Oil')
```

[60]:
```
50
```

[61]:
```
print(stock_prices)
```

```
{'Tesla': 240, 'Apple': 220, 'Microsoft': 135}
```

[64]:
```
stock_prices_hist = {
    'Tesla' : [238, 239, 240],
    'Apple' : [222, 220, 220, 218],
    'Microsoft': [136, 134, 135, 135, 133, 136]
}
```

[65]:
```
print(stock_prices_hist)
```

```
{'Tesla': [238, 239, 240], 'Apple': [222, 220, 220, 218], 'Microsoft': [136,
134, 135, 135, 133, 136]}
```

[73]:
```
for k, v in stock_prices_hist.items():
    print('The starting price of ' + k + ' was: ' + str(v[0]))
    print('The closing price of ' + k + ' was: ' + str(v[-1]))
```

```
    print('The average price of ' + k + ' was: ' + str(sum(v)/len(v)))
    print('')
```

```
The starting price of Tesla was: 238
The closing price of Tesla was: 240
The average price of Tesla was: 239.0

The starting price of Apple was: 222
The closing price of Apple was: 218
The average price of Apple was: 220.0

The starting price of Microsoft was: 136
The closing price of Microsoft was: 136
The average price of Microsoft was: 134.83333333333334
```

4 Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

4.0.1 Defining a Function

- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

[75]:
```
def print_hello_world():
    print("Hello World!")
```

[76]:
```
print_hello_world
```

[76]:
```
<function __main__.print_hello_world()>
```

[77]:
```
type(print_hello_world)
```

[77]:
```
function
```

[78]:
```
print_hello_world()
```

```
Hello World!
```

[81]:
```
def print_hello(name):
    print("Hello " + name + "!")
```

[82]:
```
print_hello('Karoly')
```

```
Hello Karoly!
```

[85]:
```
print_hello(3)
```

```
                                                                            ␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-85-0153d678d8a4> in <module>
----> 1 print_hello(3)


<ipython-input-81-907d9a5aba4f> in print_hello(name)
      1 def print_hello(name):
----> 2     print("Hello " + name + "!")


TypeError: must be str, not int
```

[87]:
```
def print_hello(val):
    print("Hello " + str(val) + "!")
```

[89]:
```
print_hello('Karoly')
```

```
Hello Karoly!
```

[91]:
```
print_hello(7)
```

```
Hello 7!
```

[97]:
```
def my_sum(L):
    s = 0
    for elem in L:
        s = s + elem
    return(s)
```

[106]:
```
my_sum([2,7,3])
```

[106]: 12

[99]:
```
def avg(L):
    s = 0
    c = 0
    for elem in L:
        s = s + elem
        c = c + 1
    a = s / c
    return(a)
```

[107]:
```
avg([2,7,3])
```

[107]: 4.0

[109]:
```
def avg2(L):
    return(sum(L)/len(L))
```

[111]:
```
avg2([2,7,3])
```

[111]: 4.0

[114]:
```
def my_pow(base, power):
    prod = 1
    for i in range(power):
        prod = prod * base
    return(prod)
```

[117]:
```
my_pow(2,4)
```

[117]: 16

[123]:
```
def intersect(dict1,dict2):
    in_both = []
    for k in dict1.keys():
        if k in dict2.keys():
            in_both = in_both + [k]
    return(in_both)
```

[124]:
```
account1 = {
    'USD' : 10000,
    'EUR' : 5000,
    'GBP' : 400
    }
account2 = {
    'USD' : 5000,
    'HUF' : 500000,
    'EUR' : 2000
    }
intersect(account1, account2)
```

[124]:
```
['USD', 'EUR']
```

5 Homework Week 2

5.0.1 Problem 1:

Write a function that returns the minimum positive value for a list input. `def min_pos(L):`

Eg.: `min_pos([-2, 1, 3, 0]) = 1`

5.0.2 Problem 2:

Given two dictionaries, can be expected as a `{key string:int value}`, return a new dictionary with the merged dictionary which contains keys from both dictionaries but adds up values for multiple occurences. Eg. "'python account1 = { 'USD' : 10000, 'EUR' : 5000, 'GBP' : 400 }

account2 = { 'USD' : 5000, 'HUF' : 500000, 'EUR' : 2000 }

merge_accounts(account1, account2) = { 'USD' : 15000, 'EUR' : 7000, 'GBP' : 400, 'HUF' : 500000 }"'