Columbia Business School
AT THE VERY CENTER OF BUSINESS

Introduction to Python

Session 4

Recall: Data Structures

Boolean (Bool)
String (Str)
Integer (Int)
Float (Float)

What you've learned

List (List)
Tuple (Tuple)

Dictionary (Dict)

Today!

+ More challenging examples

Finding the Type of a Variable

```
type(variable_name)
```

Limitations of Lists

```
stocks = ['FB', 'AAPL', 'NFLX']
prices  = [341.37, 133.11, 527.07]
```

These information are linked. Can we store them so that they are associated with each other for faster data retrieval?

Columbia Business School
AT THE VERY CENTER OF BUSINESS

Dictionary

Dictionary

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

created with curly brackets

"key": value structure

- key - string/ number
- value - string/ number/ list/ dictionary, almost anything!

Another way to think about dictionary

```
d = {'FB': 341.37,
     'AAPL': 133.11,
     'NFLX': 527.07}
```

key	value
'FB'	341.37
'AAPL'	133.11
'NFLX'	527.07

labels do not need to be lined up, but makes it easier to read

Accessing the values

Put the key inside square brackets to access its value (similar to list)

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
print(d['FB'])
```

```
341.37
```

Accessing the values

What happens if the key is not present?

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
print(d['MSFT'])
```

```
KeyError
```

Accessing the values

A safer way to access values without throwing an error

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
print(d.get('MSFT'))
```

```
None
```

no error raised if key doesn't exist

```
print(d.get('AAPL'))
```

```
133.11
```

Reassigning values

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
d['NFLX'] = 529.49
print(d)
```

similar to lists

```
{'FB': 341.37, 'AAPL': 133.11, 'NFLX': 529.49}
```

Reassigning values

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

useful when incorporating in loops

```
d['NFLX'] += 2.42
print(d)
```

works for lists too!

```
{'FB': 341.37, 'AAPL': 133.11, 'NFLX': 529.49}
```

Adding key/value pair

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
d['MSFT'] = 268.02
print(d)
```

```
{'FB': 341.37, 'AAPL': 133.11, 'NFLX': 527.07, 'MSFT': 268.02}
```

Removing key/value pair

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07, 'MSFT': 268.02}
```

```
del d['MSFT']
print(d)
```

```
{'FB': 341.37, 'AAPL': 133.11, 'NFLX': 527.07}
```

Loops

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
for stock in d:
    print(stock)
```

loops over keys by default

```
FB
AAPL
NFLX
```

Looping over keys

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
for stock in d.keys():
    print(stock)
```

```
FB
AAPL
NFLX
```

iterable collection of all the keys

Looping over values

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
for price in d.values():
    print(price)
```

```
341.37
133.11
527.07
```

iterable collection of all the values

Try it!

You work at a trading company that usually stores its day's trading data as two separate lists.

```
ticker = ['AMZN', 'GOOG', 'TSLA']
price = [3420.70, 2533.49, 682.70]
```

Having learned about the benefits of dictionaries, you propose storing the data as a single dictionary instead for faster data retrieval. Using your knowledge of lists and loops, write code that builds such a dictionary.

Try it!

Does your same code still work if you executed one more trade?

```
ticker = ['AMZN', 'GOOG', 'TSLA', 'NVDA']
price = [3420.70, 2533.49, 682.70, 800.30]
```

We aim to write code that works well generally given similar input data.

Solution

allows code to work for lists of different lengths

```
result = {}
for i in range(len(ticker)):
    result[ticker[i]] = price[i]
print(result)
```

```
{'AMZN': 3420.7, 'GOOG': 2533.49, 'TSLA': 682.7}
```

Logic Operators

```
d = {'FB': 341.37, 'AAPL': 133.11,'NFLX': 527.07}
```

```
print('FB' in d)
```

```
True
```

Booleans are useful for if/else statements

```
print('AMZN' in d)
```

```
False
```

Try it!

Suppose someone sits on a trading floor, and every time they hear a stock being traded, they write down the stock's ticker. The result is one long string, containing a list of tickers separated by spaces.

Write some code that take this string and returns a dictionary in which each key is a ticker, and each value is the number of times the ticker was traded.

```
tickers = "GOOG MSFT MSFT GOOG MSFT MSFT TSLA
PLTR PLTR PLTR GOOG GOOG GOOG"
```

Buggy Solution

```
split_tickers = tickers.split()
ticker_dict = {}
for ticker in split_tickers:
    ticker_dict[ticker] += 1
print(ticker_dict)
```

```
KeyError
```

`ticker_dict` does not contain `"GOOG"` key

Solution

```
split_tickers = tickers.split()
ticker_dict = {}
for ticker in split_tickers:
    if ticker in ticker_dict:
        ticker_dict[ticker] += 1
    else:
        ticker_dict[ticker] = 1
print(ticker_dict)
```

```
{'GOOG': 5, 'MSFT': 4, 'TSLA': 1, 'PLTR': 3}
```

Exercises

The waiver exam will have a 1.5 star difficulty on average.

Try it! ★☆☆

As a project manager, you need to oversee the expenditure for each project to ensure that each project can be fully executed without exceeding the budget. You have two lists, one containing expenditure on procurement for each project and the other containing expenditure on marketing. However, you would like to monitor the total expenditure. Create a new list that sums the element together at the same index and then sort the resulting list in descending order.

```
procurement = [212, 646, 718]
marketing = [323, 213, 714]
```

Solution

```
result = list()
for i in range(len(procurement)):
    total = procurement[i] + marketing[i]
    result.append(total)
result.sort(reverse=True)
print(result)
```

```
[1432, 859, 535]
```

Try it! ★★☆

123

The Fibonacci sequence is a very famous sequence of numbers named after Italian mathematician Fibonacci. One reason for its popularity is its connection to the golden ratio.

The first two terms of the sequence are 0 and 1, and each successive term is the sum of the two preceding terms.

Write Python code that uses loops to print the terms of this sequence. Stop when you reach numbers larger than 5,000

Solution


```
n1 = 0
n2 = 1
while n1 < 5000:
    this_number = n1 + n2
    n1 = n2
    n2 = this_number
    print(n1)
```

Try it! ★☆☆

You recently obtained a list of your clients in no specific order from your manager. To help decide which clients to focus on, your manager marked the important clients by prefacing their names with a letter “x”. For example, “daniel” would become “xdaniel”. Produce two lists - one that contains high priority clients, and one that contains clients that are not prioritized

```
clients = ['george', 'xchristina', 'meredith', 'izzie',
'xmiranda', 'derek']
```

should return

```
high_priority_clients = ['xchristina', 'xmiranda']
low_priority_clients = ['george', 'meredith', 'izzie', 'derek']
```

Solution

```
high_priority_clients = []
low_priority_clients= []
for s in clients:
    if s[0] == "x":
        high_priority_clients.append(s)
    else:
        low_priority_clients.append(s)
print(high_priority_clients)
print(low_priority_clients)
```

```
['xchristina', 'xmiranda']
['george', 'meredith', 'izzie', 'derek']
```

Try it- Let's Go Further

You recently obtained a list of your clients in no specific order from your manager. To help decide which clients to focus on, your manager marked the important clients by prefacing their names with a letter "x". For example, "daniel" would become "xdaniel". Sort the list of clients alphabetically, except group all clients' name that begin with x first to determine the order in which to serve them.

```
clients = ['george', 'xchristina', 'meredith', 'izzie',
'xmiranda', 'derek']
```

should return

```
['xchristina', 'xmiranda', 'derek', 'george', 'izzie',
'meredith']
```

Solution

```
high_priority_clients.sort()
low_priority_clients.sort()
all_clients = high_priority_clients + low_priority_clients
print(all_clients)
```

```
['xchristina', 'xmiranda', 'derek', 'george',
'izzie', 'meredith']
```

Try it! ★★☆

Your company sells hundreds of products, each with its own unique product IDs. Each ID can be of any length and is made up of numbers and letters only.

```
id = ['a1b2c3d4', 'ewry7edh0d', '13n2f3pm42i9']
```

Your manager wants to revamp product IDs for easier identification. She proposes creating a new product ID that contains all of the numbers in the ID in their original order of appearance, followed by all of the letters in the string in their original order of appearance. Write Python code to assist your manager with this request. The input id provided should return `1234abcd, 70ewryedhd, 1323429nfpmi`.

Solution

```
for i in range(len(product_id)):
    nums = ''
    letters = ''
    current_word = product_id[i]
    for letter in current_word:
        if letter.isalpha():
            letters += letter
        if letter.isdigit():
            nums += letter
    print(nums + letters)
```

```
1234abcd
70ewryedhd
1323429nfpmi
```

Try it! ★★☆

Recall the USPS example we covered in our very first class. Now, we want to compile all addresses in the same state together for faster deliveries.

```
list_of_addresses = ['3022 Broadway, New York, NY 10027',
                     '123 W 83rd St, New York, NY 10025',
                     '1200 E. California Blvd., Pasadena, CA 91125',
                     '20 W 34th St, New York, NY 10001',
                     '155 W 105th St, New York, NY 10025',
                     '1600 Holloway Ave, San Francisco, CA 94132']
```

Given this list of addresses, create two lists - one containing all the states in these addresses, and one containing all the zip codes in these addresses. CHALLENGE: Could you make it so that no state/zip is repeated?

Solution

```
states = []
zip_codes = []
for address in list_of_addresses:
    state = address.split(',')[-1].split()[0]
    zip_code = address.split(',')[-1].split()[1]
    if state not in states:
        states.append(state)
    if zip_code not in zip_codes:
        zip_codes.append(zip_code)
print(states)
print(zip_codes)
```

```
['NY', 'CA']
['10027', '10025', '91125', '10001', '94132']
```

Alternative Solution

```
states = []
zip_codes = []
for address in list_of_addresses:
    state = address[-8:-6]
    zip_code = address[-5:]
    if state not in states:
        states.append(state)
    if zip_code not in zip_codes:
        zip_codes.append(zip_code)
print(states)
print(zip_codes)
```

since addresses are properly formatted

```
['NY', 'CA']
['10027', '10025', '91125', '10001', '94132']
```

Try it! ★★☆

Let's try creating a dictionary to sort these address by state. Every entry in the dictionary should be a state, and the content of that entry should be the list of addresses in that state.

```
list of addresses = ['3022 Broadway, New York, NY 10027',
                     '123 W 83rd St, New York, NY 10025',
                     '1200 E. California Blvd., Pasadena, CA 91125',
                     '20 W 34th St, New York, NY 10001',
                     '155 W 105th St, New York, NY 10025',
                     '1600 Holloway Ave, San Francisco, CA 94132']
```

Solution

```
dict_of_addresses = {}
for address in list_of_addresses:
    state = address.split(',')[-1].split()[0]
    if state in dict_of_addresses:
        dict_of_addresses[state].append(address)
    else:
        dict_of_addresses[state] = [address]
print(dict_of_addresses)
```

```
{'NY': ['3022 Broadway, New York, NY 10027', '123 W 83rd St,
New York, NY 10025', '20 W 34th St, New York, NY 10001', '155
W 105th St, New York, NY 10025'], 'CA': ['1200 E. California
Blvd., Pasadena, CA 91125', '1600 Holloway Ave, San
Francisco, CA 94132']}
```