Flexible Indices: Dict

Practice of the Day (http://www.cs.virginia.edu/~up3f/cs1110/practice-of-the-day/) 13:

```
# Convert the following while loop into a for loop that produces the same output

i = 7
while i > -3:
   print(i*i)
   i -= 1

for i in range(7, -3, -1):
   print (i*i)
```

POTD 14:

```
# Write a function that takes a sentence,
# check if there are repeated words.
# Your program will return a list of the repeated words
# Note: repeated words are words that occur multiple times in consecutive
# multiple occurrences of the same word should be reported once
#
# If there is no repeated word or the file is empty,
# return []
#
# For example, if a sentence contains
#      'I will do more more practice and my bring my my my questions to class'
#
# Your program will return
#      ['more', 'my']
#
# Note that your choice of data structure can affect the return formats.
# For this practice, any formats would be fine.
#
# You should consider:
# - What data structure would you be using?
# - Why would you use that data structure?
#


def check_repeated_words(sentence):
   result = []
   list = sentence.split()
   for i in range(len(list) - 1):
       if list[i] == list[i+1]:
           if list[i] not in result:
               result.append(list[i])
   return result


print(check_repeated_words('I will do more more practice and my bring my my my questions
to class'))
```

Dictionary: complex data type (collection), mutable (individual items can be accessed and changed), unordered (no index)

Each item in a dictionary has a **value** and a **key**

(Each item in a list or string has a **value** and an **index**)

Keys are "deterministic", meaning they refer to only one value

For example, in SIS, we have our Student ID as a **key** (vlb9ae) and our name as a **value** (Layne Berry)

Keys do not need to be integers incremented by 1, as indexes do

To create a dictionary, supply both a key and a value, separated by a colon

```
phonebook = {'george':'111-1111', 'tom':'222-2222'}
print(phonebook)
print(phonebook['tom'])
```

Must access values by their key; some values can have multiple keys, but no key ever has multiple values

Assigning a new value to an existing key deletes the old value

```
phonebook['george'] = '333-3333'
print(phonebook)
```

Assigning a new value to a nonexistent key adds the new key and value to the dictionary

```
phonebook['joe'] = '444-4444'
print(phonebook)
```

Delete an element from a dictionary two ways (**del** and **dict.pop()**)

```
del phonebook['tom']
# removes 'tom':'222-2222'
phone_number = phonebook.pop('tom')
# returns '222-2222' as phone_number
# and removes 'tom':'222-2222'
```

Delete all elements with **dict.clear()**

Get the length of a dictionary the same way as other collections

```
num_items = len(phonebook)
print(phonebook)
```

Retrieve data from a dictionary

```
phonebook.get('joe')
# retrieves the value at 'joe' (444-4444)
# if 'joe' doesn't exist, returns "none"
phonebook.get('tim', 'not available')
# if 'tim' exists, retrieves its value
# otherwise, returns "not available"
phonebook.items()
# retrieves everything
phonebook.keys()
# retrieves just the keys
phonebook.values()
# retrieves just the values
```

Final note: all data types can be in a dictionary, and **in** and **not in** work on dictionaries

In-Class Examples in Pycharm:

```
phonebook = {'upsorn':'111-1111'}
print(phonebook)
# {'upsorn':'111-1111'}
print('upsorn' in phonebook.keys())
# True
print('luther' in phonebook.keys())
# False
phonebook['luther'] = '222-2222'
phonebook['craig'] = '333-3333'
print("keys =", phonebook.keys())
print("values =", phonebook.values())
print("items =", phonebook.items())
# not printing data in any particular order
print(phonebook.clear())
# None
print(phonebook)
# {}
phonebook['upsorn'] = '111-1111'
phonebook['craig'] = '333-3333'
print(phonebook)
# {'upsorn': '111-1111', 'craig': '333-3333'}
phonebook['upsorn'] = '222-2222'
print(phonebook)
# {'upsorn': '222-2222', 'craig': '333-3333'}
```

Exercise from Slides my answer:

```
experience = {}

def add_experience(times):
    global experience
    for i in range(times):
        place = input("Where did you work? ")
        year = int(input("What year were you working there? "))
        job = input("What was your position? ")
        experience[year] = [job, place]

times = int(input("How many jobs have you worked?"))
add_experience(times)
print(experience)
```

Exercise from Slides class answer:

```
def record_experience():
    experience = {}
    more_input = 'y'
    while(more_input == "y"):
        company = input("Enter company: ")
        year = input("Enter year: ")
        name = input("Enter experience: ")
```

```
        experience[year] = [name, company]
        more_input = input("Do you want to enter more? (y/n) ")
    return experience
print(record_experience())
```