Lecture 3 - Prepared

January 23, 2023

1 Outline

- Dictionary
- Boolean data type and conditional statement

[1]:
```
# Recall a list
fruit_list = ['apple', 'grape', 'banana']
print(fruit_list[0])
# Recall a tuple
fruit_tuple = ('apple','grape','banana')
print(fruit_tuple[0])
```

```
apple
apple
```

2 Dictionary

A dictionary in Python is a collection of **key-value pairs**, where each **key is unique**. Dictionaries are enclosed in curly braces {} and values can be assigned and accessed using square braces []

[2]:
```
# Creating a dictionary
fruit_dict ={'apple': 2, 'banana':4,'grape':'2 lbs'}
print(fruit_dict)
# Retrieving an element
print('Quantity for apple', fruit_dict['apple'])
print('Quantity for grape', fruit_dict['grape'])
```

```
{'apple': 2, 'banana': 4, 'grape': '2 lbs'}
Quantity for apple 2
Quantity for grape 2 lbs
```

[3]:
```
# add an item to dictionary
fruit_dict["orange"]=10
print(fruit_dict)
#updating an element
fruit_dict["orange"]=5
print(fruit_dict)
```

```
# delete an item
del fruit_dict['banana']
print(fruit_dict)
```

```
{'apple': 2, 'banana': 4, 'grape': '2 lbs', 'orange': 10}
{'apple': 2, 'banana': 4, 'grape': '2 lbs', 'orange': 5}
{'apple': 2, 'grape': '2 lbs', 'orange': 5}
```

[4]:
```
# Iterating over dictionary
for key in fruit_dict:
    print(key, fruit_dict[key])
```

```
apple 2
grape 2 lbs
orange 5
```

3 Important Remarks

- Dictionaries are unordered, meaning that the items have no index and are not stored in a particular order.
- The **keys** in a dictionary must be **immutable**. They can strings, numbers, or tuples.
- The values can be of any data type.

[5]:
```
#Try creating a dictonary
# Play with some errors
new_dict = {(2,2): 'A', 'origin':'O'}
```

4 Important built-in functions

- keys(): returns a list of all the keys in the dictionary
- values(): returns a list of all the values in the dictionary
- items(): returns a list of all the key-value pairs in the dictionary as tuples
- get(key): returns the value of the key passed as an argument, returns **None** if key is not present
- pop(key): removes the key-value pair and returns the value of the key passed as an argument, raises an error if key is not present
- clear(): removes all key-value pairs from the dictionary

Practice problem 1. Recall that we can multiply a list and a tuple (by a constant n). Can we do it for a dictionary? why? 2. Use the above built-in functions to play around with your dictionary.

[6]:
```
#Creating list
dict1 = {'math':['310','275','183'],'physics':[110]}

# Using keys()
print(dict1.keys())

# Using values()
print(dict1.values())

# Using items()
print(dict1.items())

# Using get()
print(dict1.get('math'))
print(dict1.get('physics'))

# Using pop()
print(dict1.pop('physics'))
print(dict1)

# Using clear()
dict1.clear()
print(dict1)
```

```
dict_keys(['math', 'physics'])
dict_values([['310', '275', '183'], [110]])
dict_items([('math', ['310', '275', '183']), ('physics', [110])])
['310', '275', '183']
[110]
[110]
{'math': ['310', '275', '183']}
{}
```

5 Boolean Data Type

In Python, a boolean is a data type that can have one of two values: **True** or **False**. Boolean values are commonly used to represent the truth or falsehood of an expression, statement, or condition.

[7]:
```
# Assigning value to a Boolean variable
x = True
y = False
#Boolean values can also be the result of comparison operators
# like ==, !=, <, >, <=, and >=
a = 10
b = 5
z = 10 >= 5
print(z)
```

```
True
```

[8]:
```
# Boolean values can also be the result of logical operators like and, or, not.␣
↪These operators are used to combine and negate boolean expressions
x = 5
y = 10
z = 15
print((x < y) and (y < z)) # Output: True
print(not (x > y)) # Output: True
```

```
True
True
```

6 Basic logic

- The statement "x and y" returns **True** if both x, y are **True**. Otherwise, it returns **False**.
- The statement "x or y" returns **True** if either x or y is **True**. Otherwise, it returns **False**.
- The 'not x' operator negates the boolean value of x.

Boolean values and expressions are also used in control flow statements like **if, else, elif** to control the execution of a program.

7 Conditional Statements

- In Python, a conditional statement is used to check if a certain condition is true or false, and then execute a block of code based on the outcome of the condition.
- The most commonly used conditional statements are "if", "if-else", and "if-elif-else".

7.1 If statement

The basic syntax for an "if" statement is as follows:

```
if condition:
    # code to be executed if condition is True
```

[9]:
```
x = 5
if x > 0:
    print("x is positive")
```

```
x is positive
```

7.2 If-else statement

The basic syntax for an "if-else" statement is as follows:

```
if condition:
    # code to be executed if condition is True
```

```
else:
    # code to be executed if condition is False
```

[10]:
```
x = -5
if x > 0:
    print("x is positive")
else:
    print("x is non-positive")
```

```
x is non-positive
```

7.3 If-elif-else statement

The basic syntax for an "if-elif-else" statement is as follows:

```
if condition1:
    # code to be executed if condition1 is True
elif condition2:
    # code to be executed if condition1 is False and condition2 is True
else:
    # code to be executed if condition1 and condition2 are False
```

[11]:
```
x = 5
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

```
x is positive
```

[12]:
```
# It's also possible to use logical operators like and, or, not in the␣
 ↪conditions
x = 5
y = 6
if x > 0 and y > 0:
    print("x and y are both positive")
```

```
x and y are both positive
```

8 Check out ticket

Problem: Given a list of integers, create a new dictionary that stores the frequency of each integer in the list.

Hint: need to define a function

```
def count_frequency(nums):
    ...
```

Expect:

```
nums = [1, 2, 3, 2, 4, 1, 5, 2, 1]
count_frequency(nums)
# Output: {1: 3, 2: 3, 3: 1, 4: 1, 5: 1}
```