COMP10001

WORKSHOP #5

Shevon Mendis
shevonm@unimelb.edu.au

dicts

- Similar to how real-world dictionaries store (unique) words and their meanings, a `dict` is a data structure that stores an (unordered) collection of key-value pairs
  - The keys are unique and **must be of an immutable type**
  - The values associated with the keys can be of any type
- Useful when we need to store information specific to different objects in our code
- Defining a new, empty dictionary in Python:
  - `dictionary = {}`
  - `dictionary = dict()`

`dict` Operations

- Accessing a value associated with a particular key:
  - `dict[key]`
- Accessing the keys:
  - `dict.keys()`
- Accessing the values:
  - `dict.values()`
- Accessing all the key-value pairs:
  - `dict.items()`
- Make a copy:
  - `dict.copy()`
  - **Note:** This makes a *shallow* copy

dict Operations

- Adding a new key:
  - `dictionary[new_key] = associated_value`
- Updating a key with a new value:
  - `dictionary[key_to_update] = new_value`
- Removing a single key-value pair:
  - If you want to use the value:
    - `dictionary.pop(key[, default])`
  - Otherwise:
    - `del dictionary[key]`
- Removing all the key-value pairs:
  - `dictionary.clear()`

sets

- Essentially represents a mathematical set, ie. a data structure that stores an (unordered) collection of unique items.
    - Elements of a set **must be of an immutable type**

- Useful when we to store **unique** data (or remove duplicates)

- Defining a new, empty set in Python:
    - `my_set = set()`

`set` Operations

- Adding a new element:
  - `set.add(new_elem)`
  - Since all the elements must be unique, adding an element that already exists will have no effect

- Removing an element:
  - To remove and retrieve a random element:
    - `set.pop()`
  - To remove a specific element:
    - `set.remove(elem_to_remove)`

`set` Operations

- Intersection: returns the elements common to both sets
  - `set1.intersect(set2)` *or* `set1 & set2`

- Union: returns a new set containing all the unique elements from both sets
  - `set1.union(set2)` *or* `set1 | set2`

- Set Difference: returns a new set containing the elements in the first set that aren't in the second set (**Note:** The ordering of the sets matter)
  - `set1.difference(set2)` *or* `set1 - set2`

- Other useful methods include `copy()`, `clear()`, `issubset()`

Mutability

- `lists` are **mutable**- once we define it, we **can** change its contents

```
>>> my_list = [1, 2, 3]
>>> my_list[2] = 7
>>> my_list
[1, 2, 7]
```

- `tuples` are **immutable**- once defined, we **can't** change its contents

```
>>> my_tuple = (1, 2, 3)
>>> my_tuple[2] = 7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Mutability

- *However*, it's actually the *bindings* that are unchangeable, not the objects they are bound to. In other words, if the `tuple` contains a mutable object, we **can** change *that* object

```
>>> my_tuple = ([1, 2, 3], 14)
>>> my_tuple[0][1] = 9
>>> my_tuple
([1, 9, 3], 14)
```

Mutability

- Mutable data types:
  - `list`
  - `set`
  - `dict`

- Immutable data types:
  - `bool`
  - `int`
  - `float`
  - `str`
  - `tuple`

None

- The `None` keyword is used to define a null value (ie. a value that doesn’t exist)
    - **NOT** the same as `0`, `False`, `""`, `[]`, etc.

- `None` is a data type of its own- **`NoneType`**
    - Comparing `None` to anything, except `None` itself, will return `False`

- All variables assigned to `None` point to the **same** (null) object in memory

None in the real world

1. None is the default return object when a function terminates without returning something

```python
def is_even(num):
    if num % 2 == 0:
        print("Even")
    else:
        print("Odd")
```

```
>>> result = is_even(2)
Even
>>> print(result)
None
```

None in the real world

2. None can be used to initialise the value that we might not have found yet

```python
def get_highest_scorer(marks):
    highest_mark   = 0
    highest_scorer = None

    for scorer, mark in marks.items():
        if mark > highest_mark:
            highest_mark   = mark
            highest_scorer = scorer

    return highest_scorer
```

```
>>> marks = { "Troy": 64, "Gabriella": 99, "Chad": 63, "Sharpei": 70 }
>>> get_highest_scorer(marks)
'Gabriella'
```

None in the real world

3. None can be used as a default parameter

```
def bad_function(new_elem, starter_list=[]):
    starter_list.append(new_elem)
    return starter_list
```

```
>>> bad_function('a')
['a']
>>> bad_function('b')
['a', 'b']
>>> bad_function('c')
['a', 'b', 'c']
```

Source: https://realpython.com/null-in-python/#using-none-as-a-default-parameter

None in the real world

3. None can be used as a default parameter

```
def good_function(new_elem, starter_list=None):
     if starter_list == None:
          starter_list = []
      starter_list.append(new_elem)
      return starter_list
```

```
>>> good_function('a')
['a']
>>> good_function('b')
['b']
>>> good_function('c')
['c']
```

Source: https://realpython.com/null-in-python/#using-none-as-a-default-parameter

`sorted()` vs `.sort()`

- `sorted()` is a **function** that takes a collection as input and returns a **new** list of sorted elements

- `.sort()` is a `list` **method** that sorts a list *in-place*, ie. it *mutates* the **original** list
  - As it mutates the original list, it returns `None`

Namespaces

- **Namespace:** A mapping from names (of variables or functions) to objects. It defines the collection of variables which can be used in a certain part of your program

- **Global namespace:** The *global* namespace is the collection of variables and functions available outside of any function in a program

- **Local namespace:** When a function is called, it will have a *local* namespace, which is unique to that function's execution and forgotten once it terminates

- **Scope:** The area of a program where a particular namespace is used
  - Variables in a function's *local* namespace are said to be in the function's scope
  - Python looks in the most local namespace first, and if it can't be found there, proceeds to check the global namespace