DICTIONARIES

What? W3n?

DICTIONARY

- **Dictionary** is a collection of **key-value** pairs.
- This establishes a connection between data points and their identifiers (keys) for easier access and manipulation.
- Dictionaries are mutable, which means you can add, modify, or remove key-value pairs after the dictionary is created.
- **Keys** must be of an immutable data type, such as strings, numbers.

HOW TO CREATE A DICTIONARY

```
>>> person = {
      "name": "John",
      "age": 30,
      "city": "New York"
      }
>>> person
{'name': 'John', 'age': 30, 'city': 'New York'}
>>> person['age'] = 31 # Modify a value
>>> person
{'name': 'John', 'age': 31, 'city': 'New York'}
```

HOW TO ITERATE OVER A DICTIONARY?

```
my_dict = {"a": 1, "b": 2, "c": 3}

# Iterate over the keys using a for loop
for key in my_dict:
    print(key) # prints the keys: a, b, c

for key in my_dict:
    print(my_dict[key]) # prints the values: 1, 2, 3
```

LIST VS DICTIONARY

- LIST does not care about the data inside, you just add to it
  - Finding data is based on ordering, e.g. get entry[2] or looping over all entries and getting the values where names[i] == “Sarah”
  - It is fast to add to list, but searching is dependent on data amount and is slower
- DICTIONARY
  - Data is added as key-value pairs
  - Any data can be retrieved based on the key, for example, ages[“Sarah”] or inhabitants[“Thailand”]
  - Adding & searching always takes a certain default amount of time.

USING DICTIONARY

- Example: W10E01.py

USING DICTIONARY

- We can change the values in a dictionary
- phone_numbers[“Elon Musk”] = “555-3928372”

- We can also add new entries with the same method
- phone_numbers[“Bill Gates”] = “555-39482949”

DICTIONARY METHODS

- The dictionary has quite similar methods as a List
  - .pop(key) # Remove the item with the key name and return it
  - .popitem() # Removes the last item and returns it
  - .clear() # Clears the entire dictionary
- There is special function for sorting Dictionaries
  - sorted(dictionary, key=key, reverse=reverse) # reverse the order, key is a function on how to reverse

LISTS WITH DICTIONARIES

- Usually we have dictionaries defining complex data and list containing entries
- cars=[{ “brand”: “Ford”, ”year:” 2015, “price”: 15000},
  { “brand”: “Skoda”, ”year:” 2012, “price”: 5000}]
- for car in cars:
  if car[“brand”] == “Ford”:
  print(”Year of the Ford is:”, car[“year”]

SHALLOW COPY VS. DEEPCOPY

- A shallow copy is a copy of an object that is not a complete, independent copy.
- Instead, it creates a new object, but the new object references the same "nested elements" as the original object.
- This has importance, if the original list or dictionary contains mutable elements as their "nested elements".
- `deepcopy()` is a function of the `copy` module that creates a deep copy of an object.
- **A deep copy** is a new object that is a completely independent copy of the original object, including all of its elements or nested objects, and their elements, and so on.

SHALLOW COPY / EXAMPLE

See example: W10E03.py