

Beginner's Python Cheat Sheet - Dictionaries

What are dictionaries?

Python's dictionaries allow you to connect pieces of related information. Each piece of information in a dictionary is stored as a key-value pair. When you provide a key, Python returns the value associated with that key. You can loop through all the key-value pairs, all the keys, or all the values.

Defining a dictionary

Use curly braces to define a dictionary. Use colons to connect keys and values, and use commas to separate individual key-value pairs.

Making a dictionary

```
alien_0 = {'color': 'green', 'points': 5}
```

Accessing values

To access the value associated with an individual key give the name of the dictionary and then place the key in a set of square brackets. If the key you provided is not in the dictionary, an error will occur.

You can also use the `get()` method, which returns `None` instead of an error if the key doesn't exist. You can also specify a default value to use if the key is not in the dictionary.

Getting the value associated with a key

```
alien_0 = {'color': 'green', 'points': 5}
```

```
print(alien_0['color'])  
print(alien_0['points'])
```

Getting the value with `get()`

```
alien_0 = {'color': 'green'}
```

```
alien_color = alien_0.get('color')  
alien_points = alien_0.get('points', 0)  
alien_speed = alien_0.get('speed')
```

```
print(alien_color)  
print(alien_points)  
print(alien_speed)
```

Adding new key-value pairs

You can store as many key-value pairs as you want in a dictionary, until your computer runs out of memory. To add a new key-value pair to an existing dictionary give the name of the dictionary and the new key in square brackets, and set it equal to the new value.

This also allows you to start with an empty dictionary and add key-value pairs as they become relevant.

Adding a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}
```

```
alien_0['x'] = 0  
alien_0['y'] = 25  
alien_0['speed'] = 1.5
```

Starting with an empty dictionary

```
alien_0 = {}  
alien_0['color'] = 'green'  
alien_0['points'] = 5
```

Modifying values

You can modify the value associated with any key in a dictionary. To do so give the name of the dictionary and the key in square brackets, then provide the new value for that key.

Modifying values in a dictionary

```
alien_0 = {'color': 'green', 'points': 5}  
print(alien_0)
```

```
# Change the alien's color and point value.  
alien_0['color'] = 'yellow'  
alien_0['points'] = 10  
print(alien_0)
```

Removing key-value pairs

You can remove any key-value pair you want from a dictionary. To do this use the `del` keyword and the dictionary name, followed by the key in square brackets. This will delete the key and its associated value.

Deleting a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}  
print(alien_0)
```

```
del alien_0['points']  
print(alien_0)
```

Visualizing dictionaries

Try running some of these examples on pythontutor.com, and then run one of your own programs that uses dictionaries.

Looping through a dictionary

You can loop through a dictionary in three ways: you can loop through all the key-value pairs, all the keys, or all the values.

Dictionaries keep track of the order in which key-value pairs are added. If you want to process the information in a different order, you can sort the keys in your loop, using the `sorted()` function.

Looping through all key-value pairs

```
# Store people's favorite languages.  
fav_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',  
}
```

```
# Show each person's favorite language.  
for name, language in fav_languages.items():  
    print(f"{name}: {language}")
```

Looping through all the keys

```
# Show everyone who's taken the survey.  
for name in fav_languages.keys():  
    print(name)
```

Looping through all the values

```
# Show all the languages that have been chosen.  
for language in fav_languages.values():  
    print(language)
```

Looping through all the keys in reverse order

```
# Show each person's favorite language,  
# in reverse order by the person's name.  
for name in sorted(fav_languages.keys(),  
                  reverse=True):  
    language = fav_languages[name]  
    print(f"{name}: {language}")
```

Dictionary length

You can find the number of key-value pairs in a dictionary using the `len()` function.

Finding a dictionary's length

```
num_responses = len(fav_languages)
```

Python Crash Course

A Hands-on, Project-Based
Introduction to Programming

ehmatthes.github.io/pcc_3e



Nesting - A list of dictionaries

It's sometimes useful to store a number of dictionaries in a list; this is called nesting.

Storing dictionaries in a list

```
# Start with an empty list.
users = []

# Make a new user, and add them to the list.
new_user = {
    'last': 'fermi',
    'first': 'enrico',
    'username': 'efermi',
}
users.append(new_user)

# Make another new user, and add them as well.
new_user = {
    'last': 'curie',
    'first': 'marie',
    'username': 'mcurie',
}
users.append(new_user)

# Show all information about each user.
print("User summary:")
for user_dict in users:
    for k, v in user_dict.items():
        print(f"{k}: {v}")
    print("\n")
```

You can also define a list of dictionaries directly, without using append():

```
# Define a list of users, where each user
# is represented by a dictionary.
users = [
    {
        'last': 'fermi',
        'first': 'enrico',
        'username': 'efermi',
    },
    {
        'last': 'curie',
        'first': 'marie',
        'username': 'mcurie',
    },
]

# Show all information about each user.
print("User summary:")
for user_dict in users:
    for k, v in user_dict.items():
        print(f"{k}: {v}")
    print("\n")
```

Nesting - Lists in a dictionary

Storing a list inside a dictionary allows you to associate more than one value with each key.

Storing lists in a dictionary

```
# Store multiple languages for each person.
fav_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
}

# Show all responses for each person.
for name, langs in fav_languages.items():
    print(f"{name}: ")
    for lang in langs:
        print(f"- {lang}")
```

Nesting - A dictionary of dictionaries

You can store a dictionary inside another dictionary. In this case each value associated with a key is itself a dictionary.

Storing dictionaries in a dictionary

```
users = {
    'einstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
    },

    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
    },
}

for username, user_dict in users.items():
    full_name = f"{user_dict['first']} "
    full_name += user_dict['last']

    location = user_dict['location']

    print(f"\nUsername: {username}")
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

Levels of nesting

Nesting is extremely useful in certain situations. However, be aware of making your code overly complex. If you're nesting items much deeper than what you see here there are probably simpler ways of managing your data, such as using classes.

Dictionary Comprehensions

A comprehension is a compact way of generating a dictionary, similar to a list comprehension. To make a dictionary comprehension, define an expression for the key-value pairs you want to make. Then write a for loop to generate the values that will feed into this expression.

The zip() function matches each item in one list to each item in a second list. It can be used to make a dictionary from two lists.

Using a loop to make a dictionary

```
squares = {}
for x in range(5):
    squares[x] = x**2
```

Using a dictionary comprehension

```
squares = {x:x**2 for x in range(5)}
```

Using zip() to make a dictionary

```
group_1 = ['kai', 'abe', 'ada', 'gus', 'zoe']
group_2 = ['jen', 'eva', 'dan', 'isa', 'meg']

pairings = {name:name_2
             for name, name_2 in zip(group_1, group_2)}
```

Generating a million dictionaries

You can use a loop to generate a large number of dictionaries efficiently, if all the dictionaries start out with similar data.

A million aliens

```
aliens = []

# Make a million green aliens, worth 5 points
# each. Have them all start in one row.
for alien_num in range(1_000_000):
    new_alien = {
        'color': 'green',
        'points': 5,
        'x': 20 * alien_num,
        'y': 0
    }

    aliens.append(new_alien)

# Prove the list contains a million aliens.
num aliens = len(aliens)

print("Number of aliens created:")
print(num_aliens)
```

Weekly posts about all things Python

mostlypython.substack.com

