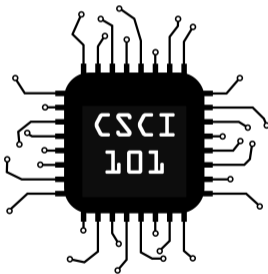


Dictionaries

A Key-Value Relationship



Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is ?

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991
- `langs["C"]` is ?

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991
- `langs["C"]` is 1972

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991
- `langs["C"]` is 1972
- `langs[1995]` is ?

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991
- `langs["C"]` is 1972
- `langs[1995]` is **Error**
 - This results in a **KeyError** exception

Dictionaries

Python Dictionaries are a one-way key-value mapping. They are like a list, but elements are accessed using a key, rather than a numerical index.

```
langs = {"Python": 1991, "C": 1972, "Java": 1995}
```

Access is similar to a list, but the key replaces the offset:

- `langs["Python"]` is 1991
- `langs["C"]` is 1972
- `langs[1995]` is **Error**
 - This results in a **KeyError** exception

How can we add to a dictionary?

Suppose we wanted to add the FORTRAN language:

```
langs["FORTRAN"] = 1957
```

Consider Dictionaries Like a Table

Having trouble with dictionaries? Think of them like a table, where the **key** is the column you look up an entry by, and the **value** is the column you are looking for.

Name (key)	Phone No. (value)
Alice	(123) 456-7890
Bill	(212) 555-1212
Jane	(444) 555-6666
Mary	(890) 123-4567
John	(791) 234-2255

What types of data can the **values** of a dictionary be?

The *values* of a dictionary can be of **any** type. For example, we can nest lists inside dictionaries:

```
foods = {  
    "fruits": ["oranges", "apples"],  
    "vegetables": ["broccoli", "kale"]  
}
```

What types of data can the **values** of a dictionary be?

The *values* of a dictionary can be of **any** type. For example, we can nest lists inside dictionaries:

```
foods = {  
    "fruits": ["oranges", "apples"],  
    "vegetables": ["broccoli", "kale"]  
}
```

Practice: Define the dictionary above in your interactive interpreter, then evaluate each of the following. What changes?

- 1 `foods["meats"] = ["steak", "chicken"]`
- 2 `foods["vegetables"][0] = "yum!"`
- 3 `print(len(foods))`
- 4 `print(len(foods["meats"]))`

What types of data can the **keys** of a dictionary be?

The *keys* of a dictionary can be of any **hashable** type. In other words, any data type that can be stored in a set. For example, this is **not** a valid dictionary.

```
oh_noes = {["a", "list"]: 1234}
```

Iterating over a Dictionary

Calling `.keys()` on a dictionary will give us an iterable of the keys. This allows us to loop like this:

```
systems = {"Windows NT": 1993,  
           "Linux": 1991,  
           "Mac OS X": 2001}  
for key in systems.keys():  
    print(key, systems[key])
```

```
Windows NT 1993  
Linux 1991  
Mac OS X 2001
```

The website has an example program using a dictionary as a phone book. Download it, play with it, and maybe even remix your own.

Don't forget the documentation!

Python also includes a data type for sets. A set is an unordered collection of unique elements, and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, and difference.

The *Data Structures* page in the official Python documentation has excellent information and examples on using lists, sets, and dictionaries.

These slides are nowhere near complete! Go forth and read the docs!

Here is a brief demonstration:

```
100 basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
101 print(basket)
102 # Show that duplicates have been removed
103 print(set(basket))
```