SDEV 1001

Programming Fundamentals

Dictionaries - 1

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

WE ARE ESSENTIAL TO ALBERTA

NAIT

Agenda

On the right is what we will cover today.

- Dictionary Fundamentals in Python
- Creating a Dictionary
- Accessing and Modifying Values
- Looping Through a Dictionary
- Using the get Method
- Checking for Keys
- Nested Dictionaries
- Summary

Dictionary Fundamentals in Python

What is a Dictionary?

- Dictionaries store data as key-value pairs.
- Keys are unique and usually strings; values can be any data type.
- Useful for representing structured data, like a contact or a product.
- Again this is a very common data structure in programming, that might be called a "map" or "associative array" in other languages.

Analogy to a dictionary

Think of a dictionary like a real-world dictionary where you look up a word (key) to find its definition (value). Each word is unique, just like keys in a Python dictionary.

Creating a Dictionary

Here’s the fundamental syntax for creating a dictionary in Python:

- Use curly braces `{}`.
- Separate keys and values with a colon `:`.
- Commas separate each key-value pair.

Example:

```
student = {
    "name": "Alex",
    "age": 21,
    "major": "Physics",
    "grades": [88, 92, 79]
}
```

An Important note here is that the keys need to be unique but the the values do not, and can be of any data type, including lists or other dictionaries.

So you can have dictionaries within dictionaries, or lists as values, this is a common way of representing complex data structures.

Accessing and Modifying Values

When you have a dictionary, you can access or modify its values using the keys. Note that this is different from lists, where you use indices to access elements.

How you do it:

- Use square brackets and the key to access or change a value.
- Assign a value to a new key to add it to the dictionary.

Example:

```python
print(student["name"])  # Output: Alex
student["age"] = 22     # Update value
student["email"] = "alex@example.com"  # Add new key-value pair
```

Looping Through a Dictionary

Many times you will want to loop through a dictionary to access all keys and values. You can do this using a `for` loop.

```
for key in student: # Loop through keys
    # Access value using key
    print(f"{key}: {student[key]}")
```

- This will print each key-value pair in the dictionary.

Here’s the output of the above example with the `student` dictionary:

```
name: Alex
age: 22
major: Physics
grades: [88, 92, 79]
email: alex@example.com
```

Using the get Method

If you try to access a key that doesn't exist, it will raise a `KeyError`. To avoid this, you can use the `get()` method.

- `get()` returns the value for a key, or a default if the key is missing.
- Prevents errors when accessing keys that might not exist.

```
middle_name = student.get("middle_name", "N/A")
print("Middle name:", middle_name)
```

Here the output will be:

```
Middle name: N/A
```

Because the key `"middle_name"` does not exist in the `student` dictionary, it returns the default value `"N/A"` instead of raising an error.

Checking for Keys

Sometimes you want to check if a key exists in a dictionary before accessing it. You can do this using the `in` operator.

```
if "email" in student: # Check if key exists
    print("Email is:", student["email"])
else:
    print("No email on file.")
```

This will print (because we added the email key earlier):

```
Email is: alex@example.com
```

Nested Dictionaries

Values can be other dictionaries, allowing for complex data structures.

- This allows you to represent more complex relationships, like products with multiple attributes.

```
inventory = {
    "apple": {"price": 0.5, "stock": 30},
    "banana": {"price": 0.3, "stock": 50}
}
print(inventory["apple"]["price"])  # Output: 0.5
```

This will print the price of the apple:

```
0.5
```

Summary

- Dictionaries are flexible and powerful for storing related data.
- Access, modify, and check for keys easily.
- Use `get()` and `in` for safer access.
- Nest dictionaries for more complex data.

NAIT

Example

Let’s go run a few examples together

A LEADING POLYTECHNIC COMMITTED TO YOUR SUCCESS