SDEV 1001

Programming Fundamentals

Dictionaries - 2

Expectations - What I expect from you

- No Late Assignments
- No Cheating
- Be a good classmate
- Don’t waste your time
- Show up to class

Agenda

On the right is what we will cover today.

- Looping Through Dictionaries in Python
- Looping Over Keys
- Looping Over Values
- Looping Over Key-Value Pairs with items()
- Using sorted() with Dictionaries
- Refactoring for Elegance: Example
- Summary

Looping Through Dictionaries in Python

Last class we learned about dictionaries, which are collections of key-value pairs. Today, we’ll explore how to loop through these dictionaries effectively.

Dictionaries can be looped over in several ways: by keys, values, or key-value pairs.

This is useful for tallying results, displaying data, or transforming information.

Looping Over Keys

The default way to loop through a dictionary is by its keys. You can also explicitly use the `.keys()` method, which returns all the keys in the dictionary as an array-like object.

```
grades = {"Alice": 85, "Bob": 92, "Charlie": 78}
print("Students:")
for student in grades.keys():
    print("-", student)
```

The above will output:

```
Students:
- Alice
- Bob
- Charlie
```

Looping Over Values

The inverse of looping over keys is to loop over the values. You can use the `.values()` method to get all the values in the dictionary. Note these are great for calculations or aggregations.

```python
grades = {"Alice": 85, "Bob": 92, "Charlie": 78}
total = 0
for score in grades.values():
    total += score
print("Average score:", total / len(grades))
```

Thhe above will output:

```
Average score: 85.0
```

Looping Over Key-Value Pairs with items()

Another great way to loop through a dictionary is to access both keys and values simultaneously using the `.items()` method. This returns each key-value pair as a tuple, which can be unpacked in the loop.

```python
grades = {"Alice": 85, "Bob": 92, "Charlie": 78}
for name, score in grades.items():
    print(f"{name} scored {score}")
```

The above will output:

```
Alice scored 85
Bob scored 92
Charlie scored 78
```

Note: Remember that tuples are like lists but are immutable, meaning you cannot change their contents after creation.

Using sorted() with Dictionaries

You can also sort the output of a dictionary when looping through it. This is particularly useful for displaying results in a specific order, such as alphabetically by key.

```python
inventory = {"apples": 5, "bananas": 2, "oranges": 8}
for fruit, count in sorted(inventory.items()):
    print(f"{fruit.title()}: {count}")
```

The above will output:

```
Apples: 5
Bananas: 2
Oranges: 8
```

Refactoring for Elegance: Example

Here’s a handy example of how to refactor code to make it more elegant and efficient when working with dictionaries.

```python
inventory = {"apples": 0, "bananas": 0, "oranges": 0}
purchases = ["apples", "bananas", "apples", "oranges", "apples"]
for item in purchases:
    inventory[item] += 1

print("Inventory:")
for fruit, count in sorted(inventory.items()):
    print(f"{fruit.title()}: {count}")
```

This will output:

```
Apples: 3
Bananas: 1
Oranges: 1
```

Summary

- Loop through dictionaries with `.keys()`, `.values()`, and `.items()`.
- Use `sorted()` for ordered output.
- Refactor code to use direct key access for clarity and efficiency.
- Tuples are often used when unpacking key-value pairs in loops.

NAIT

Example

Let’s go run a few examples together