intro to dictionaries (class slides)

CSc 110 - dictionaries

Announcements

- Last lab for short project 06 is Today – no lab for the rest of the week, we re-start lab sessions Monday March 11

Dictionaries

Data Structure

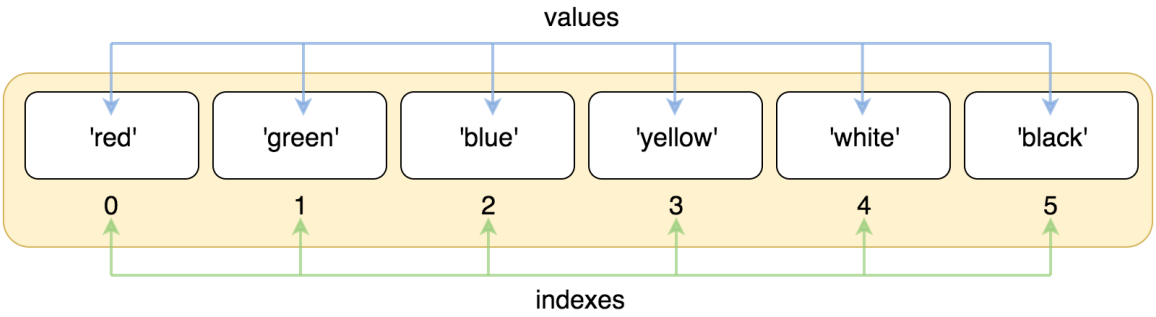
- **A data-structure is a way of arranging and organizing data in a computer program**
- Python has several useful data-structures built into the language
  - One is a **list** (already covered)
  - Another, **dictionary**

Mapping

- Many data structures allow data to be stored and retrieved using a concept called **mapping**
- **Mapping** is the process of associating one value with another (a **key** with a **value**)
  - Sometimes also referred to as Hashing or Associativity

Mapping

- Lists map **keys** to **values** too!
  - Indices **of** the list are the **keys**
  - Elements **in** the list are the **values**
- Keys (indices) are used to acess or modify the elements in the list



Mapping and Lists

```
numbers = [12, 49, -2, 26, 5, 17, -6]
```

- What are the keys?
- What are the values?
- Which keys map to which values?

Mapping and Lists

```
numbers = [12, 49, -2, 26, 5, 17, -6]

# Using the key 3 to lookup the associated value of 26
# and then save the value into variable
new = numbers[3]

# Modifying the list so that the key 5 now maps to 77
# instead of 17
numbers[5] = 77
```

Dictionary

- Like lists:
  - Associates a set of keys to their corresponding values
  - Each key has exactly 1 associated value
- Unlike lists:
  - The keys can be types other than ints: strings

Dictionary

Example (mapping strings to integers)

```
players = { "Lebron James": 38,
            "Steph Curry":  35,
            "Devin Booker":  12 }

# Using the key "Lebron James"
# to lookup the number 38
players["Lebron James"]

# Modifying the number associated with
# "Devin Booker" from 12 to 26
players["Devin Booker"] = 26
```

Evaluate the expressions

```
word_count = {"and": 324, "why": 134, "cannot": 76, "sanded": 13}
word_count["cannot"] = 90
word_count["and"] = 110
word_count["foot"] = "feet"
word_count["and"] += 10

# what will these evaluate to?
word_count["and"]
word_count["cannot"]
word_count["foot"]
```

Evaluate the expressions

```
word_count = {"and": 324, "why": 134, "cannot": 76, "sanded": 13}
word_count["cannot"] = 90
word_count["and"] = 110
word_count["foot"] = "feet"
word_count["and"] += 10

# what will these evaluate to?
word_count["and"]
```

120

```
word_count["cannot"]
```

90

```
word_count["foot"]
```

'feet'

Evaluate the expressions

```
num_to_player = {}   # A valid, but empty dictionary
num_to_player[13] = "Paul George"
num_to_player[3]  = "Chris Paul"
num_to_player[23] = "Lebron James"
num_to_player[13] = "James Harden"

# what will these evaluate to?
num_to_player[23]
num_to_player[3]
num_to_player[13]
num_to_player
```

Evaluate the expressions

```
num_to_player = {}   # A valid, but empty dictionary
num_to_player[13] = "Paul George"
num_to_player[3]  = "Chris Paul"
num_to_player[23] = "Lebron James"
num_to_player[13] = "James Harden"

# what will these evaluate to?
num_to_player[23]
```

```
'Lebron James'
```

```
num_to_player[3]
```

```
'Chris Paul'
```

```
num_to_player[13]
```

```
'James Harden'
```

```
num_to_player
```

```
{13: 'James Harden', 3: 'Chris Paul', 23: 'Lebron James'}
```

Attendance

Attendance Evaluate the expression on Gradescope.

Review: list methods

- `.append(value)`
- `.remove(value)`
- `.pop(index)`

Dictionary operations

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}

scores['A+'] = 7     # Adds a key/value pair

scores['B'] = 20     # Changes value associated with a key

scores['C']    # Retrieves a value, given a key

scores.pop('E')      # Removes a key/value pair
```

The `in` operator

With strings:

```
"a" in "aeiou"
```

```
True
```

With lists:

```
1 in [1, 2, 3, 4, 5]
```

```
True
```

And dictionary keys:

```
word_count = {"and": 324, "why": 134, "cannot": 76, "Sanded": 13}
"why" in word_count
```

```
True
```

Write a function

1. Its name is `count_vowels`
2. It takes a `string` argument
3. It creates a `dictionary`
4. It returns the `dictionary` with the count of every lowercase vowel in `string` (iterate over the string with a for loop)

```
print( count_vowels("") ) # {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
print( count_vowels("pineapple") ) # {"a": 1, "e": 2, "i": 1, "o": 0, "u": 0}
```

Write a function – solution

```
def count_vowels(string):
  counts = {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
  for i in range(len(string)):
    char = string[i]
    if char in counts:
      counts[char] += 1
  return counts

def main():
  print( count_vowels("") ) # {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
  print( count_vowels("pineapple") ) # {"a": 1, "e": 2, "i": 1, "o": 0, "u": 0}

main()
```

```
{'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}
{'a': 1, 'e': 2, 'i': 1, 'o': 0, 'u': 0}
```

Write a function

1. Its name is `count_chars`
2. It takes a `string` argument
3. It creates a `dictionary`
4. It returns the `dictionary` with the count of every characters in `string`

```
print( count_chars("") ) # {}
print( count_chars("banana") ) # {"b": 1, "a": 3, "n": 2}
```

Write a function – solution

```
def count_chars(string):
  counts = {}
  for i in range(len(string)):
    char = string[i]
```

```
    if char in counts:
      counts[char] += 1
    else:
      counts[char] = 1

  return counts

def main():
  print( count_chars("") ) # {}
  print( count_chars("banana") ) # {"b": 1, "a": 3, "n": 2}

main()
```

```
{}
{'b': 1, 'a': 3, 'n': 2}
```

Quiz 07

current time

You have 10 minutes to complete the quiz

- No need for comments, no need for a `main()`, no test cases
- Just write your function and what's inside the function
- your choice of loop (for or while)

Built-in functions you can use: `round()`, `input()`, `float()`, `str()`, `int()`, `len()` — you don't have to use all of these