ANSWERS

A3.6 Dictionaries

Data structures · A level · OCR H446 1.4.2, AQA 7517 4.2.1.4, Eduqas A500QS 1.1 · about 20 min

BugBotLab

What this lesson is about

Keys and values, dictionaries built on hash tables, and information retrieval: commands looked up by name.

- The dictionary ADT
- How it works: a hash table underneath
- Information retrieval
- A sparse map
- Look-up tables

Questions

6 marks in all

1. What does this program print? [1 mark]

```
stock = {"wheel": 4, "sensor": 2}
stock["wheel"] = stock["wheel"] - 1
stock["motor"] = 2
del stock["sensor"]
print(stock)
print("sensor" in stock, len(stock))
```

Answer:

```
{'wheel': 3, 'motor': 2}
False 2
```

wheel is changed to 3, motor is added and sensor is removed, leaving two pairs.

2. Why must the keys of a hash-table dictionary be immutable? [1 mark]

- ☐ **A** A key's hash decides its slot, so a key that changed could no longer be found
- ☐ **B** Immutable values take less memory
- ☐ **C** Values must be immutable, so keys must be too
- ☐ **D** Only numbers can be hashed

Answer: A. If the key changed, its hash would change, and a look-up would search the wrong slot.

3. Which of these can be used as keys in a Python dictionary? [1 mark]

Tick every answer that is true.

- ☐ **A** "left"
- ☐ **B** (3, 4)
- ☐ **C** [3, 4]
- ☐ **D** 7

Answer: A, B, D. Strings, tuples and numbers are immutable, so they can be hashed. A list can change, so it cannot be a key.

4. What does this program print? [1 mark]

```
counts = {}
for word in "go stop go left go".split():
    counts[word] = counts.get(word, 0) + 1
print(counts["go"], len(counts))
```

> **Answer:**
>
> ```
> 3 3
> ```
>
> go appears three times, and there are three different words: go, stop and left.

5. Which task is a dictionary best suited to? [1 mark]

- ☐ **A** Looking up a student's mark from their candidate number
- ☐ **B** Undoing edits in reverse order
- ☐ **C** Holding print jobs in the order they arrive
- ☐ **D** Storing an image's pixels row by row

> **Answer:** A. Finding a value from a unique key is what a dictionary does. The others need a stack, a queue and a 2D array.

6. A dictionary is built on a hash table. About how long does looking up one key take as the dictionary grows? [1 mark]

- ☐ **A** About the same time however many pairs there are: O(1) on average
- ☐ **B** Time proportional to the number of pairs: O(n)
- ☐ **C** Time proportional to log n, as in a binary search
- ☐ **D** Time proportional to n squared

> **Answer:** A. The key is hashed straight to its slot, so on average there is no searching at all.

The task: commands by name

`MESSAGE` is a string of words in pairs: a command, then a whole number. The known commands are: - `forward`, `backward`, `left` and `right`: the number is a distance in cm. Carry them out with `slide(dx, dy, cm)`, where `(dx, dy)` is the direction: `(0, 1)` forward, `(0, -1)` backward, `(-1, 0)` left, `(1, 0)` right. - `tone`: the number is a pitch in Hz. Play it with `tone(hz, 0.2)`. 1. Make a dictionary `DIRECTIONS` from each of the four direction words to its `(dx, dy)` tuple. Do not compare the word with each direction name: one look-up must find the direction. 2. Go through the words two at a time. Carry out a known command. For any other word print `unknown command <word>` and skip the pair. 3. Keep a second dictionary counting how many times each known command was carried out. 4. At the end, print one line per command in the order each was first carried out, as `<command>: <count>`, for example `forward: 2`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

MESSAGE = "forward 20 right 15 tone 880 forward 10 left 15 tone 660 spin 90 backward 30"

def slide(dx, dy, cm):
    """Drive cm in the direction (dx, dy): (1, 0) is right, (-1, 0) left, (0, 1) forward,
(0, -1) backward."""
    if dx > 0:
        right(50, distance=cm * dx)
    if dx < 0:
        left(50, distance=-cm * dx)
    if dy > 0:
        forward(50, distance=cm * dy)
    if dy < 0:
        backward(50, distance=-cm * dy)

words = MESSAGE.split()
```

The hint students can ask for: Split the message into words and take them two at a time. A dictionary from each direction word to its direction vector means one look-up replaces four comparisons; tone is the one command that needs its own test. A second dictionary, from word to count, keeps the tally, and a key that is not there yet starts at 0.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

MESSAGE = "forward 20 right 15 tone 880 forward 10 left 15 tone 660 spin 90 backward 30"

def slide(dx, dy, cm):
    """Drive cm in the direction (dx, dy): (1, 0) is right, (-1, 0) left, (0, 1) forward,
(0, -1) backward."""
    if dx > 0:
        right(50, distance=cm * dx)
    if dx < 0:
        left(50, distance=-cm * dx)
    if dy > 0:
        forward(50, distance=cm * dy)
    if dy < 0:
        backward(50, distance=-cm * dy)
```

```
DIRECTIONS = {"forward": (0, 1), "backward": (0, -1), "left": (-1, 0), "right": (1, 0)}
counts = {}

words = MESSAGE.split()
for i in range(0, len(words), 2):
    word = words[i]
    number = int(words[i + 1])
    if word == "tone":
        tone(number, 0.2)
    elif word in DIRECTIONS:
        dx, dy = DIRECTIONS[word]
        slide(dx, dy, number)
    else:
        print("unknown command", word)
        continue
    counts[word] = counts.get(word, 0) + 1

for word in counts:
    print(f"{word}: {counts[word]}")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.