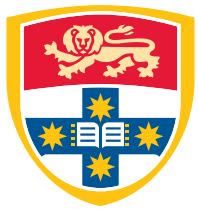
Introduction to Programming (Adv)

School of Computer Science, University of Sydney



Copyright Warning

Using Dictionaries to store key value

Dictionary data type

Dictionaries

Arrays are a contiguous area of memory

Indexed by a number

Ultimately, we want to store data and search for it.

A dictionary stores a collection of values. A **value** is added, updated, removed from the dictionary by searching for the value using a **key**.

Dictionaries are mutable

Dictionaries – creation

A dictionary can be initialised with opening and closing curly brackets { and }.

```
1  # declare an empty dictionary
2  x = { }
3  print("dictionary is " + str(x))
4
5  # declare an initialise a dictionary of 1 key:value pair
6  key = "donuts"
7  value = 3
8  y = { key:value }
9  print("dictionary y is " + str(y))
10
11 # declare an initialise a dictionary of 3 key:value pairs
12 z = { "ice cream":4, "donuts":1, "lollipop":12 }
13 print("dictionary z is " + str(z))
```

Dictionaries access

Index is the key!

```
1  store = { "ice cream":4, "donuts":1, "lollipop":12 }
2
3  print(store['ice cream'])
4  print(store['lollipop'])
5  print(store['polywaffle'])
```

```
4
12
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
KeyError: 'polywaffle'
```

Dictionaries access (cont.)

Different access with `get()` we can use the return value to check for errors

```
1  store = { "ice cream":4, "donuts":1, "lollipop":12 }
2
3  value = store.get('ice cream')
4  if value != None:
5      print(value)
6  value = store.get('lollipop')
7  if value != None:
8      print(value)
9  value = store.get('polywaffle')
10 if value != None:
11     print(value)
```

Dictionaries access (cont.)

Access all keys and or values of the dictionary using a for loop

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
2 for key in store:
3     print("key: " + key)
4
5 print()
6 for kvpair in store.items():
7     print("kvpair: " + str(kvpair))
```

```
key: ice cream
key: donuts
key: lollipop

kvpair: ('ice cream', 4)
kvpair: ('donuts', 1)
kvpair: ('lollipop', 12)
```

dict.items() returns a collection of key value pairs as tuples

Dictionaries insertion

Setting a new item requires a new `key` and `value`

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
2 store['gum'] = 4
3 print(store)
```

```
{'ice cream': 4, 'gum': 4, 'donuts': 1, 'lollipop': 12}
```

What if that `key` already exists?!

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
2 store['donuts'] = 'homer ate them'
3 print(store)
```

```
{'ice cream': 4, 'donuts': 'homer ate them', 'lollipop': 12}
```

Dictionaries remove

Using the `del` keyword, we remove that object from the dictionary.
`del d[key]`

```
1  company = { 'sales':250000, 'legal':30000, 'promotions':90000,
2     'technical':477000, 'supplies':68000, 'taxation':120000}
3  del company['promotions']
4  print(company)
5  del company['fred']
6  print(company)
```

```
{'legal': 30000, 'sales': 250000, 'supplies': 68000, 'taxation': 120000} '
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
KeyError: 'fred'
```

Dictionaries remove (cont.)

`del` only deletes from the relevant context. In this case, within the dictionary.

```
1  x = 'The banana has legs!'
2  y = 'Abstraction is often one floor above you'
3  z = 'The old apple revels in its authority'
4  phrases = {3:x, 17:y, 9:z}
5  print(id(y))
6  print(id(phrases[17]))
7  del phrases[17]
8  print(id(y))
9  print(y)
```

```
4394962992
4394962992
4394962992
Abstraction is often one floor above you
```

It does not mean deleted object (not strictly!)

Dictionary more operations

The `in` keyword can be used to test if an object can be found within a dictionary.

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
2
3 if "ice cream" in store:
4    print("Yes! we have ice cream")
5 else:
6     print("oh no!")
```

`len` returns the number of elements in that container. In this case, the key/value pairs

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
2 print(len(store))
```

Dictionary more operations (cont.)

```
1 store = { "ice cream":4, "donuts":1, "lollipop":12 }
```

Iterating through the dictionary can be

- keys only

```
1 for k in store.keys():
2     print(k)
```

- values only

```
1 for v in store.values():
2     print(v)
```

- keys and values

```
1 for (k,v) in store.items():
2     print("key: {}\tvalue: {}".format(k,v))
```

Exercise

Write a dictionary that only accepts string based keys (any values). The dictionary capacity is 5 entries.

4 operations:

- create/initialise
- insert: key, value
- search based on key: return value
- delete based on key: (return success?)

Summary

One dictionary entry has two parts: key & value.

Powerful data structure when you want to take a *shortcut* in how you store and retrieve your data.

Hidden implementation. Unknown behaviour/performance characteristics