SMU | Lyle School of Engineering

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Sep 4 2019, Lecture 4

Agenda

- Agenda:
  - Quick review of concepts from last week
  - Dictionaries
  - Operations on dictionaries
  - Boolean expressions, logical operators
  - Control flow, if/else

Variables and Data Types

- Variables
  - containers for storing data values.
  - created the moment you first assign a value to it.

- Common data types
  - Strings: *'this is a string'*
  - Integer: *39*
  - Float: *3.1415926*
  - Boolean: *Ture, False*

Sequence types

- Tuple
  - A simple **immutable** ordered sequence of items
    - Immutable: a tuple cannot be modified once created....
    - Items can be of **mixed types**, including collection types
- List
  - **Mutable** ordered sequence of items of **mixed types**
- String
  - **Immutable**
  - Conceptually very much like a tuple

Lists

- **Mutable** ordered sequence of items of **mixed types**
- Use square brackets []
- Individual elements in the list are separated by commas
- Indexing starts from 0, not 1

```
>>> bicycles = ['trek', 'cannondale', 'redline', 'specialized']
>>> print(bicycles)
['trek', 'cannondale', 'redline', 'specialized']
>>>
```

Dictionaries

- Dictionaries store a mapping between a set of keys and a set of values
  - Keys can be any immutable type
  - Values can be any type
  - Values and keys can be of different types in a single dictionary
- You can define/add/modify/delete key-value pairs of a dictionary

```
>>> credentials = {'alice': 'this is a password', 'carl': 'this is also a password'}
>>> print(credentials['carl'])
this is also a password
>>>
```

Dictionaries

- Creating and accessing dictionaries
  - Use {} to create a dictionary (we use [] for list)
  - Specify the key-value pair if not empty

```
>>> emp_dict = {}
>>> user_info = {'name': 'ethan', 'age': 23, 'address': {'state': 'TX', 'zip': 75206}}
>>> print(user_info['name'])
ethan
>>> print(user_info['address'])
{'state': 'TX', 'zip': 75206}
>>> print(user_info['address']['zip'])
75206
>>>
```

Dictionaries

- Accessing dictionaries

```
>>> user_info = {'name': 'ethan', 'age': 23}
>>> print(user_info['age'])
23
>>> print(user_info[age])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
>>>
```

- Key error

```
>>> user_info = {'name': 'ethan', 'age': 23}
>>> print(user_info['name'])
ethan
>>> print(user_info['birthday'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'birthday'
>>>
```

Dictionaries

- Accessing dictionaries use .get() method
  - .get() method allows you to specify a default value if key not exist

```
>>> user_info = {'name': 'ethan', 'age': 23, 'employee_id': 123456}
>>> print(user_info['department'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'department'
>>> print(user_info.get('department', 'IT'))
IT
>>>
```

Dictionaries

- Adding key-value pairs to existing dictionary
  - Dictionaries are unordered
    - New entry might appear anywhere in the output

```
>>> employee_ids = {}
>>> employee_ids['alice'] = 1
>>> print(employee_ids)
{'alice': 1}
>>> employee_ids['carl'] = 2
>>> print(employee_ids)
{'alice': 1, 'carl': 2}
>>>
```

Dictionaries

- Updating dictionaries
  - Keys must be unique
  - Assigning to existing key replace its value

```
>>> employee_ids = {}
>>> employee_ids['alice'] = 1
>>> print(employee_ids)
{'alice': 1}
>>> employee_ids['carl'] = 2
>>> print(employee_ids)
{'alice': 1, 'carl': 2}
>>> employee_ids['alice'] = 3
>>> print(employee_ids)
{'alice': 3, 'carl': 2}
>>>
```

Dictionaries

- Removing dictionaries entries

```
>>> user_info = {'name': 'bob', 'password': '123456789', 'age': 23}
>>> del user_info['age']
>>> print(user_info)
{'name': 'bob', 'password': '123456789'}
>>> user_info.clear()
>>> print(user_info)
{}
>>> a = [1, 2]
>>> del a[1]
>>> print(a)
[1]
>>>
```

Dictionaries

- Useful methods

```
>>> user_info = {'name': 'carl', 'age': 23, 'employee_id': 123456}
>>> user_info.keys()
dict_keys(['name', 'age', 'employee_id'])
>>> type(user_info.keys())
<class 'dict_keys'>
>>> user_info.values()
dict_values(['carl', 23, 123456])
>>> user_info.items()
dict_items([('name', 'carl'), ('age', 23), ('employee_id', 123456)])
>>>
```

- .key() will return a list if you are using python 2.7
- use list(user_info.key()) to get a list, but it is not pythonic
  - type dict_keys is iterable

Demo

Conditional test (Boolean expression)

- Programming often involves examining a set of conditions (conditional test) and deciding which action to take based on those conditions.
- If statement allows you to examine the current state of a program and respond appropriately to that state
- Return Boolean data type: True or False

Conditional test (Boolean expression)

- Check for Equality
  - Most conditional tests compare the current value of a variable to a specific value of interest
  - Note: single = for assignment, == for check equality
  - Case sensitive

```
>>> car = 'audi'
>>> car == 'audi'
True
>>> car == 'bmw'
False
>>> car == 'Audi'
False
>>> car.upper() == 'AUDI'
True
>>> car == 'Audi'.lower()
True
>>>
```

Conditional test (Boolean expression)

- Check for Inequality
  - use !=

```
>>> requested_topping = 'mushrooms'
>>> requested_topping != 'onion'
True
>>>
```

Conditional test (Boolean expression)

- Numerical comparisons

```
>>> age = 18
>>> age == 18
True
>>> answer = 17
>>> answer != 42
True
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >=21
False
>>>
```

Conditional test (Boolean expression)

- Logical operators
  - True if a is True and b is True: a and b
  - True if a is True or b is True: a or b
  - True if a is False: not a

Conditional test (Boolean expression)

- Combine boolean expressions using logical operators

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 23
>>> age_0 >= 21 and age_1 >= 21
True
>>> (age_0 >= 21) and (age_1 >= 21)
True
>>>
```

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
>>>
```

Conditional test (Boolean expression)

- Other values are treated as equivalent to either True or False when used in conditionals:
  - False: zero, None, empty containers
  - True: non-zero numbers, non-empty objects

Conditional test (Boolean expression)

- Check whether a value is in a list

```
>>> banned_users = ['andrew', 'carolina', 'david']
>>> user = 'marie'
>>> user in banned_users
False
>>> another_user = 'david'
>>> another_user in banned_users
True
>>> user not in banned_users
True
>>> another_user not in banned_users
False
>>>
```