CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Feb 4 2020, Lecture 5

Agenda

- Agenda:
  - Quick review of concepts from last lecture
  - Functions and modules

Python loops

- if statements allow you to execute different piece of code based on the different situations (conditional test)
- Loops allow you to execute the same piece of code multiple times
- Python has two primitive loop commands
  - while loops
  - for loops

While loop

- while loop syntax

colon

indentation (4 spaces)

```
while conditional_test:
    do something
    then do something
```

- It will keep execute the code block as long as the conditional test is true.
  - usually you will need to modify the the values used in the conditional test once some conditions are met

for loop

- For-each is Python's only form of for loop, this is less like the for keyword in other programming languages.
- A for loop steps through each of the items in a collection type (list, dictionary, etc) or any other type of object which is "iterable" (remember when we call .keys() method of a dictionary)
- Often used with lists and dictionaries

colon

indentation (4 spaces)

```
for <each item> in <collection>:
    <statements>
```

while/for loops

- Using break to exit a loop
  - To exit a loop immediately without running any remaining code in the loop
- Using continue in a loop
  - Rather than breaking out of a loop entirely without executing the rest of its code, you can use the continue statement to return to the beginning of the loop based on the result of a conditional test

Avoid infinite loops

- Avoid infinite loops when using While

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

while_loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week2/while_loops.py
1
2
3
4
5

Process finished with exit code 0
```

- Might not be an issue using for loop
  - It iterates through each element in a collection (or any object that is iterable) until the end.

for loop Demo

List Comprehensions

- A powerful feature of the Python language
  - Generate a new list by applying a function to every member of an original list
  - Python programmers use list comprehensions extensively.You'll see many of them in real code

[expression for item in list]

List Comprehensions

```
li = [3, 6, 2, 7]
new_list = [elem*2 for elem in li]
print(new_list)
```

```
loops
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
[6, 12, 4, 14]

Process finished with exit code 0
```

[expression for item in list]

- Where expression is some calculation or operation acting upon the variable item.
- For each member of the list, the list comprehension
  - sets item equal to that member, and
  - calculates a new value using expression.
- It then collects these new values into a list which is the return value of the list comprehension.

Filtered List Comprehensions

[expression for item in list if filter]

- Filter determines whether expression is performed on each member of the list
- When processing each element of list, first check if it satisfies the filter condition
- If the filter condition returns False, that elements is omitted from the list before the list comprehension is evaluated.

Filtered List Comprehensions

[expression for item in list if filter]

```
li = [3, 6, 2, 7]
new_list = [elem*2 for elem in li if elem > 3]
print(new_list)

```

loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
[12, 14]

Process finished with exit code 0
```

Nested List Comprehensions

- Since list comprehensions take a list as input and produce a list as output, they are easily nested:

- Self-test: what do you think the nested_li will be?

```
li = [3, 2, 4, 1]
nested_li = [elem*2 for elem in
 [item+1 for item in li]]
```

Nested List Comprehensions

```python
li = [3, 2, 4, 1]
nested_li = [elem*2 for elem in
 [item+1 for item in li]]

print(nested_li)
```

loops

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/loops.py
[8, 6, 10, 4]

Process finished with exit code 0
```

- The inner comprehension produces: [4, 3, 5, 2]
- So, the outer one produces: [8, 6, 10, 4]

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding

Feb 6 2020, Lecture 6

Agenda

- Agenda:
  - Quick review of concepts from last lecture
  - Functions and modules

Functions

- A function
  - A block of code which only runs when it is called
  - One way to organize and reuse code
  - You can pass information to a function
  - You can ask a function to return data

Function definition begins with def

Function name and its arguments (optional)

```
def  get_final_answer(filename):
    """ Documentation String """
    line1
    line2
    return total_counter
...
```

Indentation(4 spaces)

Colon

return indicates the value to be sent back to the caller (optional)

First line with less indentation is considered to be outside of the function definition

Functions

- An example

```python
def greet_user():
    """Display a simple greeting."""
    print("Hello!")


greet_user()

```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello!

Process finished with exit code 0
```

Functions

- Passing information to a function

```python
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username.title() + "!")


greet_user('jesse')
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello, Jesse!

Process finished with exit code 0
```

Functions

- Arguments and Parameters
  - The variable username in the definition of greet_user() is an example of a *parameter*
  - The value "jesse" in greet_user("jesse") is an example of an *argument*

```
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username.title() + "!")


greet_user('jesse')
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello, Jesse!

Process finished with exit code 0
```

Note: the fact is sometimes people speak of parameters and arguments interchangeably.

Functions

- Passing arguments
  - A function definition can have multiple parameters, a function call may need multiple arguments
  - Ways of passing arguments
    - positional arguments
      - need to be in the same order the parameters were written
    - keyword arguments
      - where each argument consists of a variable name and a value

Functions

- Positional arguments
  - match each argument in the function call with a parameter in the function definition

```python
def get_employee_info(employee_name, employee_id):
    """Display information about an employee """
    print("Hello," + employee_name.title() + "!")
    print("Your employee id is:" + str(employee_id))


get_employee_info("jesse", 123)
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello,Jesse!
Your employee id is:123

Process finished with exit code 0
```

Functions

- Multiple function calls

```python
def get_employee_info(employee_name, employee_id):
    """Display information about an employee """
    print("Hello," + employee_name.title() + "!")
    print("Your employee id is:" + str(employee_id))


get_employee_info("jesse", 123)
get_employee_info("jake", 999)
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello,Jesse!
Your employee id is:123
Hello,Jake!
Your employee id is:999

Process finished with exit code 0
```

Functions

- Keywords arguments
  - A name-value pair that you pass to a function

```python
def get_employee_info(employee_name, employee_id):
    """Display information about an employee """
    print("Hello," + employee_name.title() + "!")
    print("Your employee id is:" + str(employee_id))


get_employee_info(employee_name="jesse", employee_id=123)
get_employee_info(employee_id=999, employee_name="jake")
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello,Jesse!
Your employee id is:123
Hello,Jake!
Your employee id is:999

Process finished with exit code 0
```

Functions

- Default Values
  - You can define a default value for each parameter, if an argument for a parameter is provided in the function call, Python uses the argument value. If not, it

```python
def get_employee_info(employee_name, employee_id=123):
    """Display information about an employee """
    print("Hello," + employee_name.title() + "!")
    print("Your employee id is:" + str(employee_id))


get_employee_info(employee_name="jesse")
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello,Jesse!
Your employee id is:123

Process finished with exit code 0
```

Functions

- Default Values
  - any parameter with a default value needs to be listed after all the parameters that don't have default values

```
def get_employee_info(employee_name="jake", employee_id):
    """Display information about an employee """
    print("Hello,"  + employee_name.title() + "!")
    print("Your employee id is:" + str(employee_id))


get_employee_info(employee_id=123)

```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
  File "/Users/xinyi/Courses/cs1340/week3/functions.py", line 1
    def get_employee_info(employee_name="jake", employee_id):
                                               ^
SyntaxError: non-default argument follows default argument

Process finished with exit code 1
```

Functions

- Equivalent Function Calls

```python
def greet_user(username, employee_id):
    """Display some simple message """
    print("hello " + username.lower())
    print("Your employee id is " + str(employee_id))


greet_user("alice", 123)
greet_user(username="alice", employee_id=123)
greet_user(employee_id=123, username="alice")
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
hello alice
Your employee id is 123
hello alice
Your employee id is 123
hello alice
Your employee id is 123

Process finished with exit code 0
```

Functions

- Avoiding argument errors

```python
def greet_user(username, employee_id):
    """Display some simple message """
    print("hello " + username.lower())
    print("Your employee id is " + str(employee_id))


greet_user("alice")
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week3/functions.py", line 7, in <module>
    greet_user("alice")
TypeError: greet_user() missing 1 required positional argument: 'employee_id'
```

```python
def greet_user(username, employee_id):
    """Display some simple message """
    print("hello " + str(username))
    print("Your employee id is " + str(employee_id))


greet_user(123, "alice")
```

greet_user()

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
hello 123
Your employee id is alice

Process finished with exit code 0
```

Functions

- Passing an arbitrary number of arguments
  - when you don't ahead of time how many arguments a function needs to accept.

```python
def make_pizza(*toppings):
    """Print the list of toppings that have been requested"""
    print(toppings)


make_pizza("pepperoni")
make_pizza("mushrooms", "green peppers", "extra cheese")
```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')

Process finished with exit code 0
```

Functions

```python
def make_pizza(username, *toppings):
    """Print the list of toppings that have been requested"""
    print("Hello " + username)
    print(toppings)


make_pizza("xinyi", "pepperoni")
make_pizza("xinyi" , "mushrooms", "green peppers", "extra cheese")

```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Hello xinyi
('pepperoni',)
Hello xinyi
('mushrooms', 'green peppers', 'extra cheese')

Process finished with exit code 0
```

```python
def make_pizza(*toppings, username):
    """Print the list of toppings that have been requested"""
    print("Hello " + username)
    print(toppings)


make_pizza("pepperoni", "xinyi")
make_pizza("mushrooms", "green peppers", "extra cheese","xinyi")

```

functions

```
/Users/xinyi/anaconda/envs/mlearn/bin/python /Users/xinyi/Courses/cs1340/week3/functions.py
Traceback (most recent call last):
  File "/Users/xinyi/Courses/cs1340/week3/functions.py", line 7, in <module>
    make_pizza("pepperoni", "xinyi")
TypeError: make_pizza() missing 1 required keyword-only argument: 'username'

Process finished with exit code 1
```

Demo

Functions

- Return values
  - A function doesn't always have to display its output directly. Instead, it can process some data and then return a value or set of values
  - The return statement takes a value from inside a function and sends it back to the line that called the function