

CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding Feb 4 2020, Lecture 5

Agenda

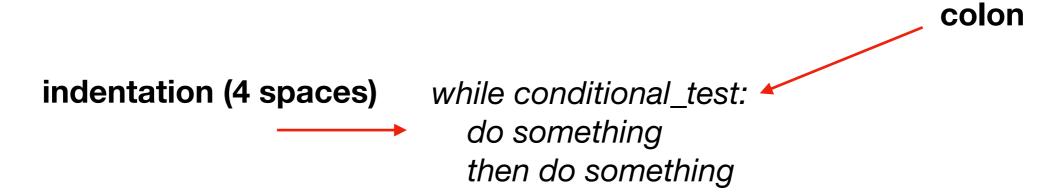
- Agenda:
 - Quick review of concepts from last lecture
 - Functions and modules

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



- 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

```
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

- 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

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



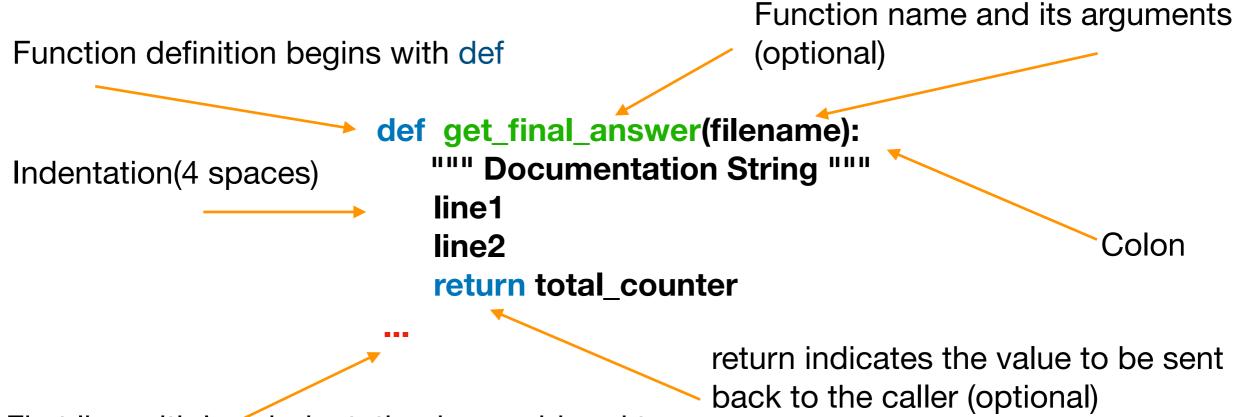
CS 1340 Introduction to Computing Concepts

Instructor: Xinyi Ding Feb 6 2020, Lecture 6

Agenda

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

- 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



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

An example

Passing information to a function

- 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.

- 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

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

```
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
```

Multiple function calls

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

```
def get_employee_info(employee_name, employee_id):
            """Display information about an employee """
 2
            print("Hello," + employee_name.title() + "!")
            print("Your employee id is:" + str(employee_id))
 6
        get_employee_info(employee_name="jesse", employee_id=123)
        get_employee_info(employee_id=999, employee_name="jake")
 8
 9
10
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
```

- 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

```
def get_employee_info(employee_name, employee_id=123):
            """Display information about an employee """
2
            print("Hello," + employee_name.title() + "!")
3
            print("Your employee id is:" + str(employee_id))
5
6
7
       get_employee_info(employee_name="jesse")
8
9
10
/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
```

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

Equivalent Function Calls

```
def greet_user(username, employee_id):
             """Display some simple message """
 2
             print("hello " + username.lower())
             print("Your employee id is " + str(employee_id))
 6
         greet_user("alice", 123)
        greet_user(username="alice", employee_id=123)
greet_user(employee_id=123, username="alice")
 8
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
```

Avoiding argument errors

```
def greet_user(username, employee_id):
           """Display some simple message """
2
           print("hello " + str(username))
3
4
           print("Your employee id is " + str(employee_id))
5
6
7
       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
```

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

```
def make_pizza(*toppings):
    """Print the list of toppings that have been requested"""
    print(toppings)
4
5
6    make_pizza("pepperoni")
7    make_pizza("mushrooms", "green peppers", "extra cheese")
8
9

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
```

```
def make_pizza(username, *toppings):
            """Print the list of toppings that have been requested"""
 2
            print("Hello " + username)
 3
            print(toppings)
 5
        make_pizza("xinyi", "pepperoni")
        make_pizza("xinyi", "mushrooms", "green peppers", "extra cheese")
 8
 9
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
```

```
def make_pizza(*toppings, username):
            """Print the list of toppings that have been requested"""
 2
            print("Hello " + username)
 3
            print(toppings)
 4
 5
 6
        make_pizza("pepperoni", "xinyi")
 7
        make_pizza("mushrooms", "green peppers", "extra cheese", "xinyi")
 8
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



- 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