Functions

David H Smith IV

University of Illinois Urbana-Champaign

Mon, Sept 28 2021

Reminders

Announcements

- **No post-reading or participation since we have a quiz.**
- Homework 7 is a review homework that is useful as practice for the quiz.
- Chromakey lab due on Sunday.
- Quiz 2 is Thursday.

More on Functions

Dynamic Typing

Python is a dynamically typed language

- The + does several different things:
  1. Integer addition
  2. Floating point addition
  3. String concatenation

Dynamic Typing

Python is a dynamically typed language

- The + does several different things:
  1. Integer addition
  2. Floating point addition
  3. String concatenation
- At **runtime** Python looks at the + operator and determines the correct behaviours based on the types or it's operands.

Dynamic Typing

Python is a dynamically typed language

- The + does several different things:
  1. Integer addition
  2. Floating point addition
  3. String concatenation
- At **runtime** Python looks at the + operator and determines the correct behaviours based on the types or it's operands.
- **Polymorphism** $\rightarrow$A single piece of code can do several things depending on the type of data it's working with.
  1. You can write less code.
  2. Can be harder to find bugs.

Poll Question: Polymorphic Functions

Which function produces an error?

```
1 def print_all(collection):
2   for item in collection:
3     print(item)
4
5 print_all({ 'k': 'v', 'CS': '105' }) #A
6 print_all(7)                         #B
7 print_all('a string')                #C
```

- A A error
- B B error
- C C error
- D A & B error
- E A & C error
- F B & C error

Scoping

Poll Question: Function Scoping

What is produced by the following code?

```
1 my_var = 11
2 def my_print(my_var):
3   print(my_var)
4
5 my_print(22)
6 print(my_var)
```

A
11
11

B
11
22

C
22
11

D
22
22

E
NameError

Poll Question: Function Scoping

What is produced by the following code?

```
1 my_var = 11
2 def change_my_var():
3   my_var = 12
4
5 change_my_var()
6 print(my_var)
```

A. 11
B. 12
C. NameError
D. None

Poll Question: Function Scoping

What is produced by the following code?

```
1 my_var = 11
2 def print_my_var():
3   print(my_var)
4
5 print_my_var()
```

A. 11
B. 12
C. NameError
D. None

Poll Question: Function Scoping

What is produced by the following code?

```
1 my_var = 11
2 def print_my_var():
3   print(my_var)
4
5 print_my_var()
```

A. 11
B. 12
C. NameError
D. None

1. We have read access but not write access in the function's scope.
2. How do we get write access to the global scope from within a function?

Poll Question: Function Scoping

What goes where the ?? is in order to (1) change the **global** value of my_var and (2) such that the user enters is printed to the screen when the code finishes running?

```
1 my_var = 11
2 def change_my_var(new_my_var):
3     ??
4
5 change_my_var(int(input("Enter a new number: ")))
6 print(my_var)
```

For this activity, have one person connect their laptop to the monitor and work on a solution as a group.

Scoping

- Every function is given a clean slate.
- Any variables written in a function are defined in the function's scope.
- The scope is destroyed when the function returns.
- If a name is read that doesn't exist in the function's scope, it tries the scope the function was defined in.

Docstrings

Docstrings!

Off to Repl.it we go. . .

Generator Functions

Generators

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 1
5 foo_gen = foo(10)
6 next(foo_gen)
```

- Generator objects: `yield` multiple values until they have finish running.

Generators

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 1
5 foo_gen = foo(10)
6 next(foo_gen)
```

A. Generator objects: `yield` multiple values until they have finish running.
B. Are defined like functions but are noteably different:
   1. They `yield` instead of return.
   2. You use `next()` to get each successive yield.
   3. Perserve their internal state until they terminate.
   4. Throws StopIteration error if you try calling next() after the generator has already finished.
   5. Almost always involve iteration (i.e., at least one for or while loop).

Generators

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 1
5 foo_gen = foo(10)
6 next(foo_gen)
```

A. Generator objects: `yield` multiple values until they have finish running.
B. Are defined like functions but are noteably different:
   1. They `yield` instead of return.
   2. You use `next()` to get each successive yield.
   3. Perserve their internal state until they terminate.
   4. Throws StopIteration error if you try calling next() after the generator has already finished.
   5. Almost always involve iteration (i.e., at least one for or while loop).
C. This is how the `enumerate()` function is implemented.

Poll Question: Function Scoping

What is produced by the following code?

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 2
5 foo_gen = foo(10)
6 a = next(foo_gen)
7 b = next(foo_gen)
8 c = next(foo_gen)
9 print(a, b, c)
```

- (A) `10 8 6`
- (B) `10 10 10`
- (C) `10 8 6 4 2`
- (D) This code contains an error

Poll Question: Function Scoping

What is produced by the following code?

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 2
5 foo_gen = foo(10)
6 foo_gen_list = list(foo_gen)
7 print(foo_gen_list)
```

A. `[10, 8, 6, 4, 2]`

B. `[10, 8, 6, 4, 2, 0]`

C. None

D. StopIteration error

Poll Question: Function Scoping

What is produced by the following code?

```
1 def foo(x):
2   while x > 0:
3     yield x
4     x -= 2
5 foo_gen = foo(10)
6 foo_gen_list = list(foo_gen)
7 print(foo_gen_list)
```

A. `[10, 8, 6, 4, 2]`
B. `[10, 8, 6, 4, 2, 0]`
C. None
D. StopIteration error

Starting to look familiar?

Poll Question: Function Scoping

Fill in the ??? in this function...

```
def my_enumerate(x):
  ???

my_enumerate_gen = my_enumerate([1, 2, 3, 4])
my_enumerate_list = list(my_enumerate_gen)
print(my_enumerate_list)
```

So that the above code behaves like this code.

```
enumerate_class = enumerate([1, 2, 3])
foo_gen_list = list(enumerate_class)
print(foo_gen_list)
```

Once you finish enumerate do the `my_range()` function so that it replicates the behaviour of the builtin `range_)` function.

General Loop Practice

Task: Validate User Input

Problem Statement: Create a function that gets 10 words that contain the letter "e", stores them in a list, then returns them. Note that this problems uses nested loops but not break or enumerate.

Task: Validate User Input

Problem Statement: Create a function that gets 10 words that contain the letter "e", stores them in a list, then returns them. Note that this problems uses nested loops but not break or enumerate.

```
1 def no_e():
2   l = []
3   for i in range(0, 10):
4     word = input("Enter a word with the letter e: ")
5     while "e" not in word:
6       word = input("Enter a word with the letter e: ")
7     l.append(word)
8   return l
```

Task: Validate User Input

Problem Statement: Create a function that keeps asking the user for strings of an even length and adding them to a list until the user enters a string of an odd length. Then return the final list. You'll want to use a "while True:" loop here.

Task: Validate User Input

Problem Statement: Create a function that keeps asking the user for strings of an even length and adding them to a list until the user enters a string of an odd length. Then return the final list. You'll want to use a "while True:" loop here.

```
1 def get_even_words():
2   l = []
3   while True:
4     user_in = input("Enter a word with an even number of vowels: ")
5     if len(user_in) % 2 != 0:
6       print("That word has an odd number of letters. Terminating!!")
7       break
8     l.append(user_in)
```

Reminders

Announcements

- **No post-reading or participation since we have a quiz.**
- Homework 7 is a review homework that is useful as practice for the quiz.
- Chromakey lab due on Sunday.
- Quiz 2 is Thursday.