Adv. Functions

David H Smith IV

University of Illinois Urbana-Champaign

Tues, Nov 16 2021

Reminders

Reminders

The folowing are due on Friday:

- **PrairieLearn:** Homework 13p2, Post-reading 14p1
- **zyBooks:** Participation 14p1

Due Monday:

- **zyBooks:** Topic 13 - Challenge Activities

Lab will be due December 3rd.

Namespaces and Scope Resolution

Namespaces, Scope, and Scope Resolution

- **Namespaces:** A mapping between names and objects.

Namespaces, Scope, and Scope Resolution

- **Namespaces:** A mapping between names and objects.
- **Scope:** The hierarchy that defines where we have access to what variables.

Namespaces, Scope, and Scope Resolution

- **Namespaces:** A mapping between names and objects.
- **Scope:** The hierarchy that defines where we have access to what variables.
- **Scope Resolution (LEGB rule):**
  1. *Local:* Things defined in a function.
  2. *Enclosed:* Things defined in a nested function.
  3. *Global:* Things defined in the program as a whole.
  4. *Built-in:* Names that are built-in to Python like `int()`.

Namespaces, Scope, and Scope Resolution

- **Namespaces:** A mapping between names and objects.
- **Scope:** The hierarchy that defines where we have access to what variables.
- **Scope Resolution (LEGB rule):**
  1. *Local:* Things defined in a function.
  2. *Enclosed:* Things defined in a nested function.
  3. *Global:* Things defined in the program as a whole.
  4. *Built-in:* Names that are built-in to Python like `int()`.
- Searches up the levels of the hierarchy.

Poll Question: Function Scope

What is printed after the following function runs?

```
x = [1, 2, 3]
def foo():
  x = []
foo()
print(x)
```

- A `[]`
- B `[1, 2, 3]`
- C NameError

Poll Question: Function Scope

What is printed after the following function runs?

```
x = [1, 2, 3]
def foo():
  x.append(4)
foo()
print(x)
```

- A [1, 2, 3, 4]
- B [1, 2, 3]
- C NameError

Poll Question: Function Scope

What is printed after the following function runs?

```
x = [1, 2, 3]
def foo():
  global x
  x = []
foo()
print(x)
```

- A `[]`
- B `[1, 2, 3]`
- C NameError

Poll Question: Function Scope

What is printed after the following function runs?

```python
x = 1
def foo():
  print("x" in locals(), end=" ")
  x = 2
  print("x" in locals(), end=" ")
foo()
```

- A `True True`
- B `True False`
- C `False True`
- D NameError

Poll Question: Function Scope

What is printed after the following function runs?

```
x = 1
def foo():
  x += 1
print(x)
foo()
print(x)
```

- A 1 2
- B 1 1
- C 2 2
- D UnboundedLocal

Poll Question: Function Scope

Why can we do this without using `global`···

```
x = [1, 2, 3]
def foo():
  x.append(4)
foo()
```

and not this without `global`?

```
x = 1
def foo():
  x -= 2
foo()
```

Poll Question: Function Scope

What is printed after the following function runs?

```
x = 1
def foo():
  print(x)
  x = 2
  print(x)
foo()
```

- A 1 2
- B 1 1
- C 2 2
- D UnboundedLocal

Default Arguments

Default Arguments: Poll Question

```
def foo(sep=",", num):
  x = []
  for i in range(num):
    x.append(str(i))
  return sep.join(x)
foo(5)
```

- A `'0,1,2,3,4'`
- B `'0,1,2,3,4,5'`
- C `'1,2,3,4,5'`
- D SyntaxError

Default Arguments: Poll Question

```
def foo(num, sep=",", mult=2):
  x = []
  for i in range(num):
    x.append(str(i * mult))
  return sep.join(x)
foo(5, mult=3, sep=".")
```

- A `'0.3.6.9.12'`
- B `'0,3,6,9,12'`
- C AttributeError
- D SyntaxError

Default Arguments: Poll Question

```
def foo(num, sep=",", mult=2):
  x = []
  for i in range(num):
    x.append(str(i * mult))
  return sep.join(x)
foo(5, ".", 3)
```

- A `'0.3.6.9.12'`
- B `'0,3,6,9,12'`
- C AttributeError
- D SyntaxError

Default Arguments: Poll Question

```
def foo(num, step=1, mult=2):
  x = []
  for i in range(num, step=step):
    x.append(str(i * mult))
  return ",".join(x)
foo(5, mult=3)
```

- A `'0,3,6,9,12'`
- B `NameError`
- C AttributeError
- D SyntaxError

Key Take Aways

Key Take Aways

1. Default arguments must follow non-default arguments (e.g., `def qux(a, b=3)`.

Key Take Aways

1. Default arguments must follow non-default arguments (e.g., `def qux(a, b=3)`.
2. You can use position to pass values in for default arguments.

Key Take Aways

1. Default arguments must follow non-default arguments (e.g., `def qux(a, b=3)`.
2. You can use position to pass values in for default arguments.
3. You can switch positions of default arguments (or arguments in general) if you use their names when calling the function.

*args

Default Arguments: Poll Question

```
def foo(*args):
  print(type(args))
```

A. list
B. tuple
C. set
D. something else?

Default Arguments: Poll Question

What is returned and printed by the function call at the bottom?

```
def foo(*things):
  x = []
  for thing in things:
    if type(thing) is int:
      x.append(thing)
  return x
print(foo(1, 2, 4.5, 3.4, "bar", "baz"))
```

A. [1, 2]
B. [1, 2, 4.5, 3.4]
C. SyntaxError
D. NameError

Default Arguments: Poll Question

What is returned by the function call at the bottom?

```
def foo(*stuff, num):
  x = []
  for thing in stuff:
    if thing % num == 0 and type(thing) is int:
      x.append(thing)
  return x
foo(5, 10, 3, "hello", "World", num=5)
```

- A `[5, 10]`
- B `[5, 10, "hello", "World"]`
- C SyntaxError
- D TypeError

Default Arguments: Poll Question

What about now?

```
def foo(*stuff, num):
  x = []
  for thing in stuff:
    if type(thing) is int and thing % num == 0:
      x.append(thing)
  return x
foo(5, 10, 3, "hello", "World", num=5)
```

- A `[5, 10]`
- B `[5, 10, "hello", "World"]`
- C SyntaxError
- D TypeError

Default Arguments: Poll Question

What is returned by the function call in the code below?

```
def foo(total, *vals):
  return total == sum(vals)
foo(15, 1, 2, 3, 4, 5)
```

A. `[5, 10]`
B. `[5, 10, "hello", "World"]`
C. SyntaxError
D. TypeError

Default Arguments

This···

```
def foo(*vals):
  return sum(vals)
foo(1, 2, 3)
```

is functionally equivalent to this···

```
def foo(vals):
  return sum(vals)
foo([1, 2, 3])
```

So arbitrary arguments are more syntactic sugar added by Python to make your code more versatile.

Default Arguments: Why?

This· · ·

```
def foo(*vals):
  total = 0
  for val in vals:
    total += val
  return total
foo(1, 2)
foo(1, 2, 3)
```

avoids the need for this. . .

```
def foo1(a,b):
  return a + b
def foo2(a,b,c):
  return a + b + c
foo1(1, 2)
foo1(1, 2, 3)
```

Key Take Aways

- The * operator is the important part. *args is only used by convention.

Key Take Aways

A. The * operator is the important part. *args is only used by convention.
B. *args can precede other parameters in the function definition however the other parameters must be called as named variables. For example:

Key Take Aways

- A The * operator is the important part. *args is only used by convention.
- B *args can precede other parameters in the function definition however the other parameters must be called as named variables. For example:

```
def foo(*vals, num):
    ...
foo(1, 2, 3, num=100)
```

vs

```
def foo(num, *vals):
    ...
foo(100, 1, 2, 3)
```

**kwargs

Default Arguments: Poll Question

What is printed out when the code below runs?

```
def foo(**kwargs):
  print(type(kwargs))
foo(a="thing", b="thing2", c=3)
```

- A list
- B tuple
- C set
- D dict

Default Arguments: Poll Question

```
def foo(**named_stuff, num):
  x = []
  for name, value in named_stuff.items():
    if value > num:
      x.append(value)
  return x
print(foo(a=10, b=4, c=1, d=15, num=5))
```

A `[10, 15]`

B `None`

C SyntaxError

D NameError

Default Arguments: Poll Question

```
def foo(**named_stuff, num):
  x = []
  for name, value in named_stuff.items():
    if value > num:
      x.append(value)
  return x
print(foo(a=10, b=4, c=1, d=15, num=5))
```

A `[10, 15]`

B `None`

C SyntaxError

D NameError

Why?

Default Arguments: Poll Question

```
def foo(num, **more_named_stuff):
  x = ""
  for key, value in more_named_stuff.items():
    if value % num == 0:
      x += key
  return x
foo(2, a=2, b=4, c=5, d=6)
```

- A 'abd'
- B TypeError
- C ValueError
- D AttributeError

Default Arguments: Poll Question

```
def foo(arg1, arg2=2, **args):
  return arg1, arg2, args
foo(5, a=1, b=2, c=3)
```

- A (5, 2, {'a': 1, 'b': 2, 'c': 3})
- B (5, 1, {'a': 1, 'b': 2, 'c': 3})
- C SyntaxError
- D NameError

Default Arguments: Poll Question

```
def foo(arg1, arg2=2, **args):
  return arg1, arg2, args
foo(5, a=1, b=2, c=3, arg2=-1)
```

- A `(5, 2, {'a': 1, 'b': 2, 'c': 3})`
- B `(5, -1, {'a': 1, 'b': 2, 'c': 3})`
- C SyntaxError
- D NameError

Key Take Aways

- As with *args, the ** operator is the important part. **kwargs is only used by convention.

Key Take Aways

- A As with *args, the ** operator is the important part. **kwargs is only used by convention.
- B **kwargs CANNOT precede other parameters in the function definition.

Key Take Aways

- A. As with *args, the ** operator is the important part. **kwargs is only used by convention.
- B. **kwargs CANNOT precede other parameters in the function definition.

```python
def foo(**vals, num):
    ...
foo(val1=1, val2=2, val3=3, num=100)
```

Key Take Aways

A. As with *args, the ** operator is the important part. **kwargs is only used by convention.

B. **kwargs CANNOT precede other parameters in the function definition.

```
def foo(**vals, num):
    ...
foo(val1=1, val2=2, val3=3, num=100)
```

C. **kwargs CANNOT precede *args:

```
def foo(num, **vals, *args):
    ...
foo(num=100, 1, 2, 3)
```

Key Take Aways

A. As with *args, the ** operator is the important part. **kwargs is only used by convention.

B. **kwargs CANNOT precede other parameters in the function definition.

```
def foo(**vals, num):
    ...
foo(val1=1, val2=2, val3=3, num=100)
```

C. **kwargs CANNOT precede *args:

```
def foo(num, **vals, *args):
    ...
foo(num=100, 1, 2, 3)
```

D. **Generally, the valid order is:** `foo(val1, val2=10, *args, **kwargs)`