ECE-203 – Programming for Engineers

Contact

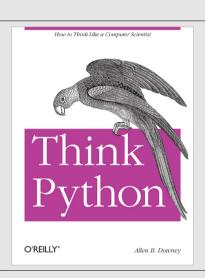
Dr. James Shackleford shack@drexel.edu Bossone 211

Office Hours: 3 – 4 pm (Tuesday)

Course Website: http://learn.dcollege.net

Textbook

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)



Grading

- 10% In-lab Programming Assignments
- 10% Take-Home Programming Assignments
- 35% Mid-term Exam
- 45% Final Exam

```
myprogram.py -- This program does blah blah blah...
 5 \text{ alpha} = 0.24
   def my_function(parameter):
                                                           module
       """ Computes the age-radius-delta product! """
 8
                                                          docstring
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
>>> import math
>>> help(math)
                                                                  module
Help on built-in module math:
                                                                 docstring
NAME
   math
FILE
    (built-in)
DESCRIPTION
    This module is always available. It provides access to the
   mathematical functions defined by the C standard.
FUNCTIONS
    acos(...)
        acos(x)
        Return the arc cosine (measured in radians) of x.
    acosh(...)
        acosh(x)
        Return the hyperbolic arc cosine (measured in radians) of x.
    asin(...)
```

```
myprogram.py -- This program does blah blah blah...
  alpha = 0.24
                                        global
                                      variables
   def my_function(parameter):
           Computes the age-radius-delta product!
 8
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blah blah blah...
                                                          function
 5 \text{ alpha} = 0.24
   def my_function(parameter):
       """ Computes the age-radius-delta product!
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blah blah blah...
 5 \text{ alpha} = 0.24
   def my_function(parameter):
       """ Computes the age-radius-delta product!
 8
       age = 34
       radius = 100
10
       color = "red"
11
                                      this stuff is global
12
       delta = parameter * alpha
13
                                      "proper" programs
14
                                         don't do this.
       return age * radius * delta
15
16
17
   result = my_function(2)
18
19
20 print result
```

```
myprogram.py -- This program does blah blah
                                                    function
                                                   signature
 5 \text{ alpha} = 0.24
   def my_function(parameter):
           Computes the age-radius-delta product!
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blah blah bla
                                                     function
                                                       body
 5 \text{ alpha} = 0.24
   def my_function(parameter):
       """ Computes the age-radius-delta product! """
 8
     age = 34
       radius = 100
10
      color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blab blab...
                                   function
 5 \text{ alpha} = 0.24
                                     name
   def my_function(parameter):
       """ computes the age-radius-delta product!
 8
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
parameter(s)
   myprogram.py -- This program does
                                             (optional)
 5 \text{ alpha} = 0.24
   def my_function(parameter):
        """ Computes the age-radius-delta product!
 8
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blah blah
                                                   function
                                                  docstring
 5 \text{ alpha} = 0.24
   def my_function(parameter):
       """ Computes the age-radius-delta product!
 8
       age = 34
       radius = 100
10
       color = "red"
11
12
13
       delta = parameter * alpha
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
>>> import math
>>> help(math)
Help on built-in module math:
NAME
   math
FILE
    (built-in)
                                                                  function
DESCRIPTION
                                                                 docstring
    This module is always available. It provides access to the
   mathematical functions defined by the C standard.
FUNCTIONS
    acos(...)
        acos(x)
       Return the arc cosine (measured in radians) of x.
    acosh(...)
        acosh(x)
        Return the hyperbolic arc cosine (measured in radians) of x.
    asin(...)
```

```
myprogram.py -- This program does blah blah blah...
 5 \text{ alpha} = 0.24
   def my_function(parameter):
       """ Computes the age-radius-delta produ
                                                      local
 8
       age = 34
                                                    variables
       radius = 100
10
       color = "red"
11
12
       delta = parameter * alpha
13
14
       return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
myprogram.py -- This program does blah blah blah...
 5 \text{ alpha} = 0.24
   def my_function(parameter):
        """ Computes the age-radius-delta product!
        age = 34
       radius = 100
10
       color = "red"
11
                                                return value
12
13
        delta = parameter * alpha
                                                (can be pretty much anything)
14
        return age * radius * delta
15
16
17
   result = my_function(2)
19
20 print result
```

```
fries = 200
   def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
 6
 8
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
11
   def my_house():
13
       apples = 10
       oranges = 23
14
       pears = 4
15
16
       print '%i apples' % apples
18
       print '%i oranges' % oranges
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
   print '%i fries' % fries
```

```
fries = 200
   def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
 8
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
   def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples'    apples
       print '%i oranges' % oranges
18
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
   print '%i fries' % fries
```

23 apples 42 burgers 21 fries

```
fries = 200
   def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
11
   def my_house():
13
       apples = 10
14
       oranges = 23
15
      pears = 4
16
17
       print '%i apples'    apples
       print '%i oranges' % oranges
18
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
   print '%i fries' % fries
```

23 apples 42 burgers 21 fries

10 apples 23 oranges 4 pears

```
fries = 200
   def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
11
   def my_house():
13
       apples = 10
14
       oranges = 23
15
      pears = 4
16
17
       print '%i apples' apples
       print '%i oranges' % oranges
18
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
   print '%i fries' % fries
```

23 apples 42 burgers 21 fries

10 apples 23 oranges 4 pears

200 fries

```
fries = 200
  def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
       apples = 10
13
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
19
       print '%i pears' % pears
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

Name search looks like this:

built-in namespace global namespace local namespace

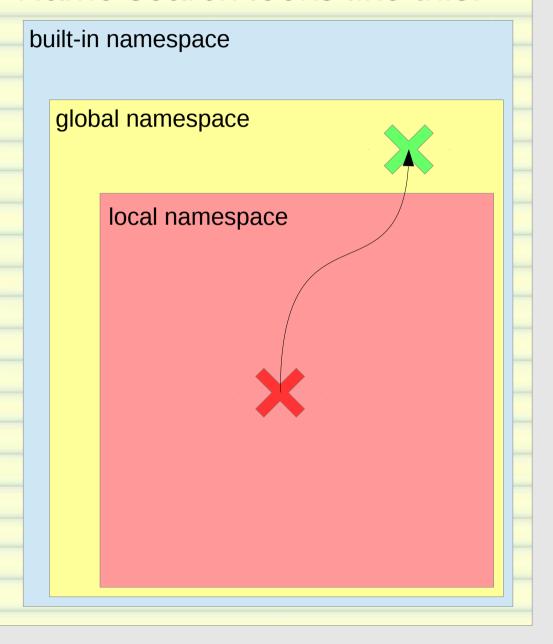
```
fries = 200
  def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
19
       print '%i pears' % pears
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

Name search looks like this:

built-in namespace global namespace local namespace

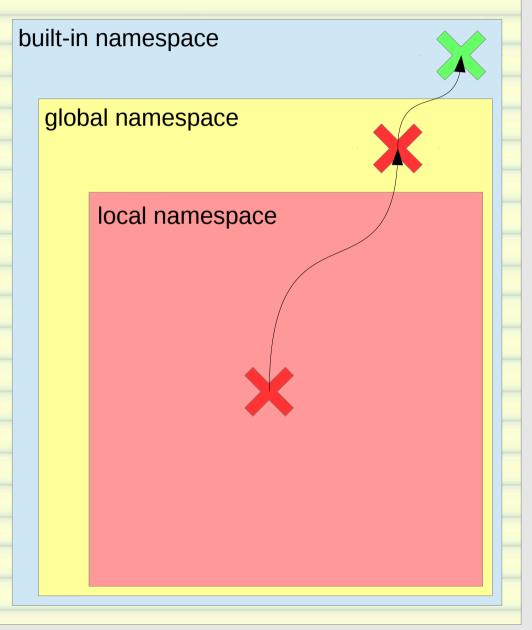
```
fries = 200
  def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
       apples = 10
13
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
19
       print '%i pears' % pears
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

Name search looks like this:



```
fries = 200
  def lunch_truck():
       apples = 23
       burgers = 42
       fries = 21
       print '%i apples' % apples
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
19
       print '%i pears' % pears
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

Name search looks like this:



```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % applés
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
                                                local namespace
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
19
       print '%i pears' % pears
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
 6
                                            global namespace
       print '%i apples' % applés
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
                                                local namespace
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
       print '%i apples' % apples
17
18
       print '%i oranges' % oranges
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                         built-in namespace
      apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % applés
       print '%i burgers' % burgers
       print '%i fries' % fries
10
                                                local namespace
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
       print '%i apples' % apples
17
18
       print '%i oranges' % oranges
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % apples
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % apples
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
17
       print '%i apples' % apples
18
       print '%i oranges' % oranges
       print '%i pears' % pears
19
20
   lunch_truck()
22
  my_house()
24
  print '%i fries' % fries
```

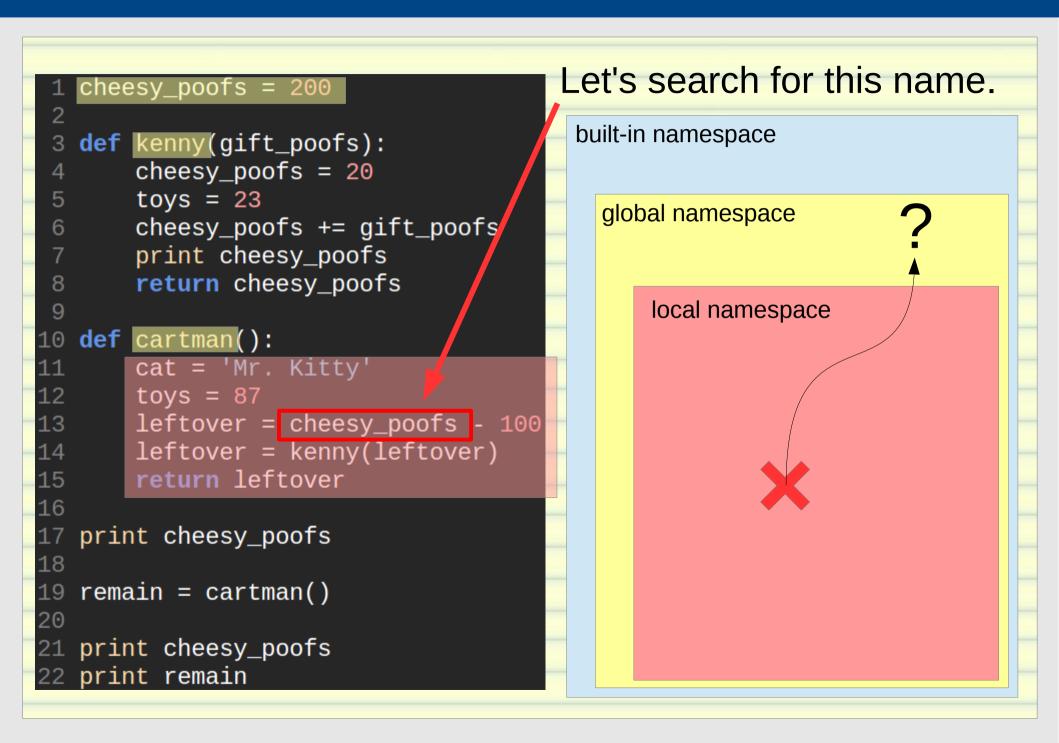
```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % apples
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
       print '%i apples' % apples
17
       print '%i oranges' % oranges
18
       print '%i prars' % pears
19
20
   lunch_truck()
  my_house()
   print '%i fries' % fries
```

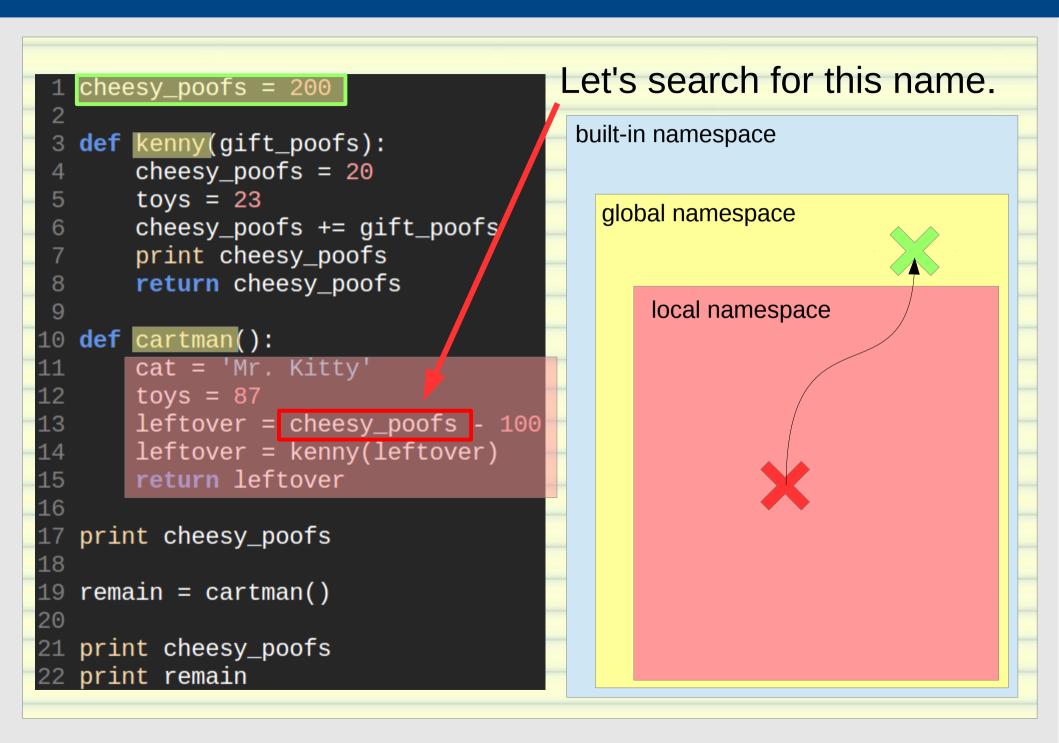
```
fries = 200
                                         Let's search for this.
  def lunch_truck():
                                          built-in namespace
       apples = 23
       burgers = 42
       fries = 21
                                            global namespace
       print '%i apples' % apples
 8
       print '%i burgers' % burgers
       print '%i fries' % fries
10
  def my_house():
13
       apples = 10
14
       oranges = 23
15
       pears = 4
16
       print '%i apples' % apples
17
       print '%i oranges' % oranges
18
       print '%i prars' % pears
19
20
   lunch_truck()
  my_house()
   print '%i fries' % fries
```

```
New Code.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                             local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
   toys = 87
12
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
     return leftover
15
16
17 print cheesy_poofs
18
19 remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
  def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
      return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs -
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```





```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       tovs = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
12
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
      return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy poofs = 20
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

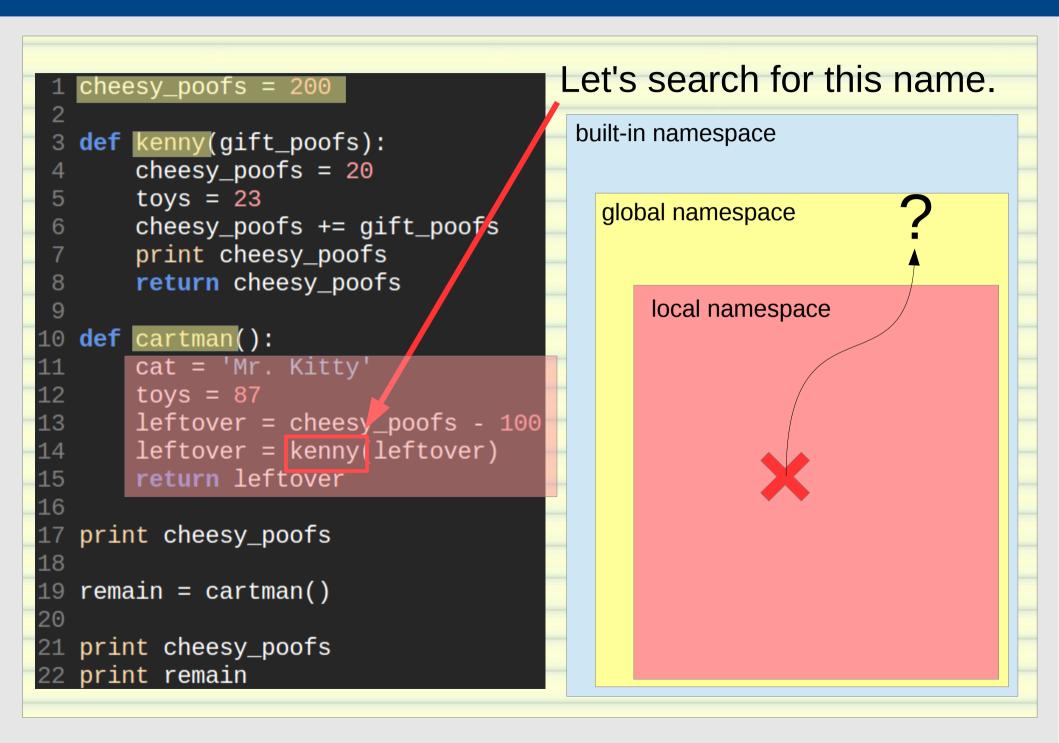
```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                             local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
   toys = 87
       leftover = cheesy_boofs - 100
13
       leftover = kenny(Zeftover)
14
      return leftover
15
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

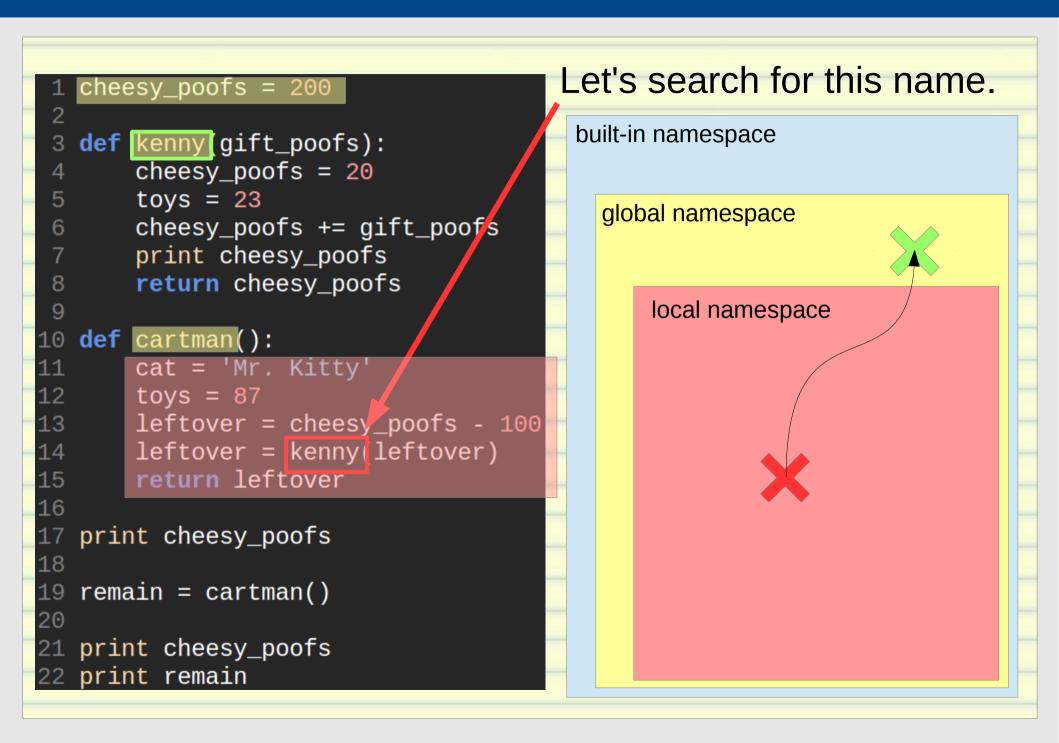
```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
   def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_boofs - 100
13
       leftover = kenny(/eftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_boofs - 100
13
       leftover = kenny(/eftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's search for this name.
   cheesy_poofs = 200
                                        built-in namespace
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                          global namespace
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                              local namespace
  def cartman():
11
       cat = 'Mr. Kitty'
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```





```
cheesy poofs = 200
                                                Print namespaces directly!
   def kenny(gift_poofs):
                                                 kenny's local namespace
        cheesy_poofs = 20
        toys = 23
        cheesy_poofs += gift_poofs
                                                 ['cheesy_poofs', 'gift_poofs', 'toys']
        print cheesy_poofs
        print dir()•
        return cheesy_poofs
10
   def cartman():
                                                 cartman's local namespace
12
        cat = 'Mr. Kitty'
13
        tovs = 87
                                                    ['cat', 'leftover', 'toys']
        leftover = cheesy_poofs - 100
14
        leftover = kenny(leftover)
15
        print dir()=
16
        return leftover
17
18
   print cheesy_poofs
                                                  global namespace
20
   remain = cartman()
22
                                                ['__builtins__', '__doc__', '__file__', '__name__', age__', 'cartman', 'cheesy_poofs', 'kenny', 'remain
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

```
cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poors
       print dir(
       return cheesy_poofs
10
   def cartman():
12
       cat = 'Mr. Kitty'
13
       tovs = 87
14
       leftover = cheesy_poofs - 100
15
       leftover = kenny(leftover)
       print dir()
16
       return leftover
17
18
19 print cheesy_poofs
20
  remain = cartman()
22
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

btw... Let's search for this name. built-in namespace global namespace local namespace

```
btw...
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poors
       print dir()
       return cheesy_poofs
10
   def cartman():
12
       cat = 'Mr. Kitty'
13
       tovs = 87
       leftover = cheesy_poofs - 100
14
       leftover = kenny(leftover)
15
       print dir()
16
       return leftover
17
18
19 print cheesy_poofs
20
21 remain = cartman()
22
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

Let's search for this name. built-in namespace global namespace local namespace

```
btw...
   cheesy poofs = 200
                                          Let's search for this name.
   def kenny(gift_poofs):
                                          built-in namespace
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poors
                                            global namespace
       print dir()
       return cheesy_poofs
10
   def cartman():
                                                local namespace
12
       \overline{cat} = 'Mr. Kitty'
13
       tovs = 87
       leftover = cheesy_poofs - 100
14
       leftover = kenny(leftover)
15
       print dir()
16
       return leftover
17
18
19 print cheesy_poofs
20
  remain = cartman()
22
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

```
btw...
   cheesy poofs = 200
                                          Let's search for this name.
   def kenny(gift_poofs):
       cheesy_poofs = 20
                                          built-in namespace
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy posts
                                            global namespace
       print dir()
       return cheesy_poofs
10
   def cartman():
                                                local namespace
12
       \overline{cat} = 'Mr. Kitty'
13
       toys = 87
       leftover = cheesy_poofs - 100
14
       leftover = kenny(leftover)
15
       print dir()
16
       return leftover
17
18
  print cheesy_poofs
20
  remain = cartman()
22
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

```
cheesy poofs = 200
                                                Print namespaces directly!
   def kenny(gift_poofs):
                                                 kenny's local namespace
        cheesy_poofs = 20
        toys = 23
        cheesy_poofs += gift_poofs
                                                 ['cheesy_poofs', 'gift_poofs', 'toys']
        print cheesy_poofs
        print dir()•
        return cheesy_poofs
10
   def cartman():
                                                 cartman's local namespace
12
        cat = 'Mr. Kitty'
13
        tovs = 87
                                                    ['cat', 'leftover', 'toys']
        leftover = cheesy_poofs - 100
14
        leftover = kenny(leftover)
15
        print dir()=
16
        return leftover
17
18
   print cheesy_poofs
                                                  global namespace
20
   remain = cartman()
22
                                                 ['__builtins__, '__doc__', '__file__', '__name__',
age__', 'cartman', 'cheesy_poofs', 'kenny', 'remain
23 print cheesy_poofs
24 print remain
25
26 print dir()
```

```
>>> dir( builtins )
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError'
  'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'E
xception', 'False', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError',
 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'KeyError', 'Keyboa
rdInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented', 'No
tImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'Referenc
eError', 'RuntimeError', 'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxEr
ror', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True', 'TypeError', '
UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'Unico
deTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivi
sionError', '_', '__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs
', 'all', 'any', 'apply', 'basestring', 'bin', 'bool', 'buffer', 'bytearray', 'bytes',
 'callable', 'chr', 'classmethod'. 'cmp', 'coerce', 'compile', 'complex', 'copyright',
 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exécfile', 'exit
', 'file', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'intern', 'isinstance', 'issubclass', 'it
er', 'len', 'license', 'list', 'locals', 'long', 'map', 'max', 'memoryview', 'min', 'n
ext', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'ra
w_input', 'reduce', 'reload', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'unichr', 'unicode',
'vars', 'xrange', 'zip']
```

```
>>> dir( builtins )
                                                                          exceptions... mostly.
 'ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError'
  'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'E
cception', 'False', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError'
 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'KeyError', 'Keyboa
 dInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented', 'No
ImplementedError', 'OSError', 'OverflowError', 'PendingDepr<u>ecationWarning'</u>. 'Referenc
Error', 'RuntimeError', 'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxEr
or', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True', 'TypeError',
JnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'Unico
deTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivi
sionError', '_', '__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs
 , 'all', 'anv', 'apply', 'basestring', 'bin', 'bool', 'buffer', 'bytearray', 'bytes',
 'callable', 'chr', 'classmethod'. 'cmp', 'coerce'. 'compile', 'complex', 'copyri<u>aht'</u>.
 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exécfile', 'exit
 , 'file', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash<mark>', 'he</mark>lp', 'hex'<mark>, 'id',</mark> 'input', 'int', 'intern', '<u>isins</u>tance', 'issubc<u>lass',</u> 'it
er', 'len', 'license', 'list', 'locals', 'lon<u>g'. 'ma</u>p', 'max', 'm<u>emoryview'. 'min</u>', 'n
ext', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'ra
w_input', 'reduce', 'reloa<u>d', '</u>repr', 'reversed'<u>, 'round</u>', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple , 'type', 'unichr', 'unicode',
 'vars', 'xrange', <mark>'</mark>zip'
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
   def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(left
14
                           200
15
      return leftover
   print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
   remain = cartman()
21 print cheesy_poofs
  print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
    cat = 'Mr. Kitty'
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

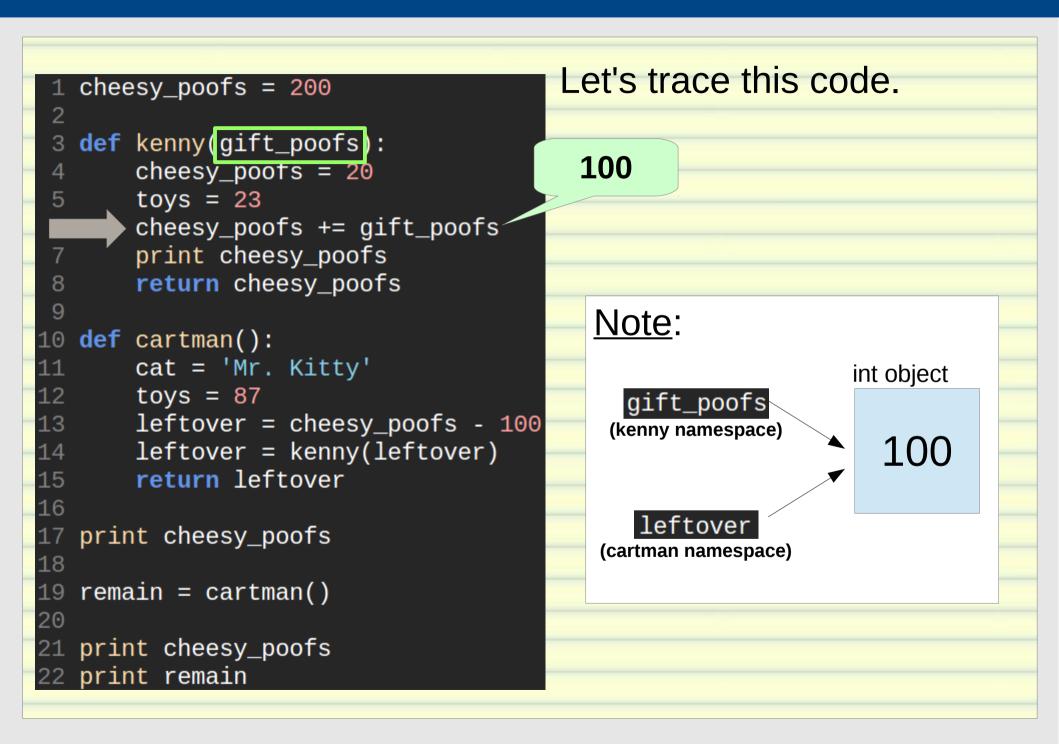
```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
       return leftover
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

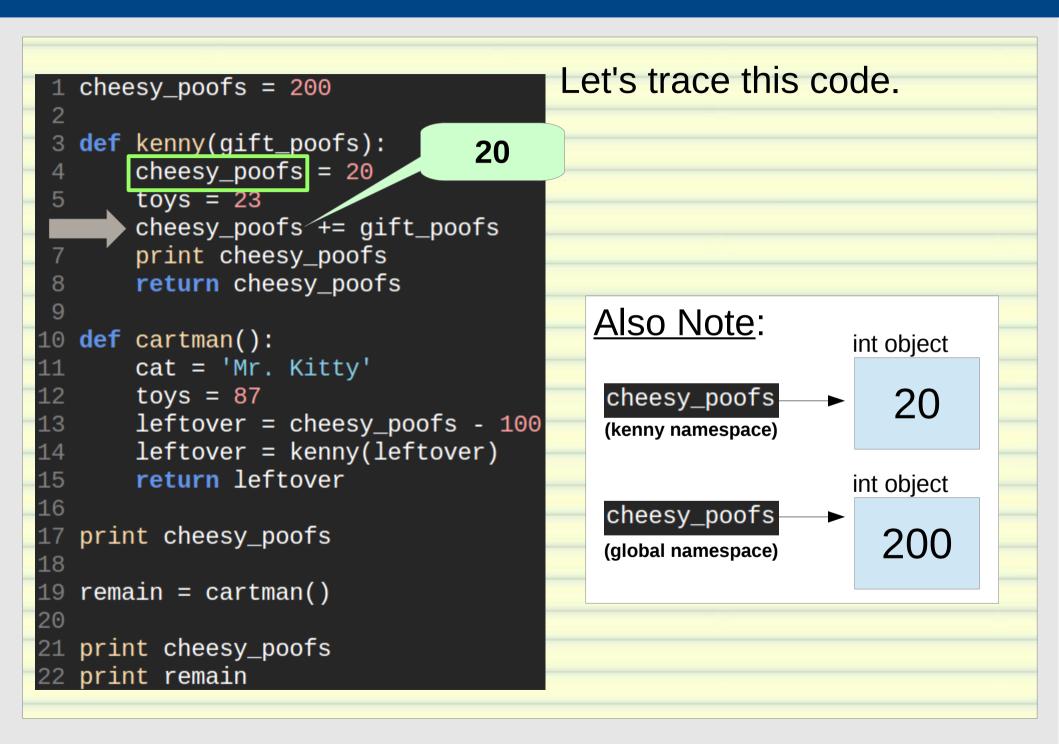
```
Let's trace this code.
  cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
                                   200
       cat = 'Mr. Kitty'
11
       toys = 87
      leftover = cheesy_poofs - 100
       leftover = kenny(leftover)
     return leftover
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
                                    100
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
       return leftover
15
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
  def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```





```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
                                120
       toys = 23
       cheesy_poofs += gift_oofs
       print cheesy_poofs/
       return cheesy_poofs
                                          Also Note:
  def cartman():
                                                              int object
       cat = 'Mr. Kitty'
11
12
       toys = 87
                                           cheesy_poofs <del>X</del> ►
                                                                 20
       leftover = cheesy_poofs - 100
13
                                           (kenny namespace)
       leftover = kenny(leftover)
14
                                                     rebind
15
      return leftover
                                                              int object
16
  print cheesy_poofs
                                                                120
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

```
cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr./ Kitty'
11
12
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
15
16
  print cheesy_poofs
18
  remain = cartman()
20
21 print cheesy_poofs
  print remain
```

Let's trace this code.

kenny's local namespace gets torn down.

['cheesy_poofs', 'gift_poofs', 'toys']



the names are destroyed.

if this results in objects with 0 references, those objects are also destroyed

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                         Note:
  def cartman():
                                                             int object
                     120
       cat = 'Mr.
11
12
       toys = 87
                                            leftover
                                                               100
       leftover = cheesy_poofs - 100
13
                                          (cartman namespace)
       leftover = kenny(leftover)
                                                    rebind
15
       return leftover
                                                             int object
16
  print cheesy_poofs
                                                               120
18
  remain = cartman()
20
21 print cheesy_poofs
22 print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
                                             cartman's local namespace
       cheesy_poofs += gift_poofs
                                                  gets torn down.
       print cheesy_poofs
                                            ['cat', 'leftover', 'toys']
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
       return leftover
  print cheesy_poofs
18
  remain´= cartman()
20
21 print cheesy_poofs
  print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
                                          Note:
   def cartman():
                                                             int object
       cat = 'Mr. Kitty'
11
12
       toys = 87
                                             let
                                                                120
       leftover = cheesy_poofs - 100
13
                                          (cartma na space)
       leftover = kenny(leftover)
14
15
      return leftover
16
                                              remain
  print cheesy_poofs
                                           (global namespace)
18
   remain = cartman()
20
21 print cheesy_poofs
   print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
                            200
18
19 remain = cartman()
   print cheesy_poofs
  print remain
```

```
Let's trace this code.
   cheesy_poofs = 200
   def kenny(gift_poofs):
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
  def cartman():
       cat = 'Mr. Kitty'
11
       toys = 87
       leftover = cheesy_poofs - 100
13
       leftover = kenny(leftover)
14
15
      return leftover
16
17 print cheesy_poofs
18
                     120
  remain = cartma
20
  print cheesy_profs
   print remain
```

```
cheesy_poofs = 200
                                        The global keyword...
                              add one
                                line
   def kenny(gift_poofs):
      global cheesy_poofs
       cheesy_poofs = 20
       toys = 23
                                                    what does it do?
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
10
11 def cartman():
       cat = 'Mr. Kitty'
       toys = 87
13
       leftover = cheesy_poofs - 100
14
       leftover = kenny(leftover)
       return leftover
16
  print cheesy_poofs
19
  remain = cartman()
21
22 print cheesy_poofs
  print remain
```

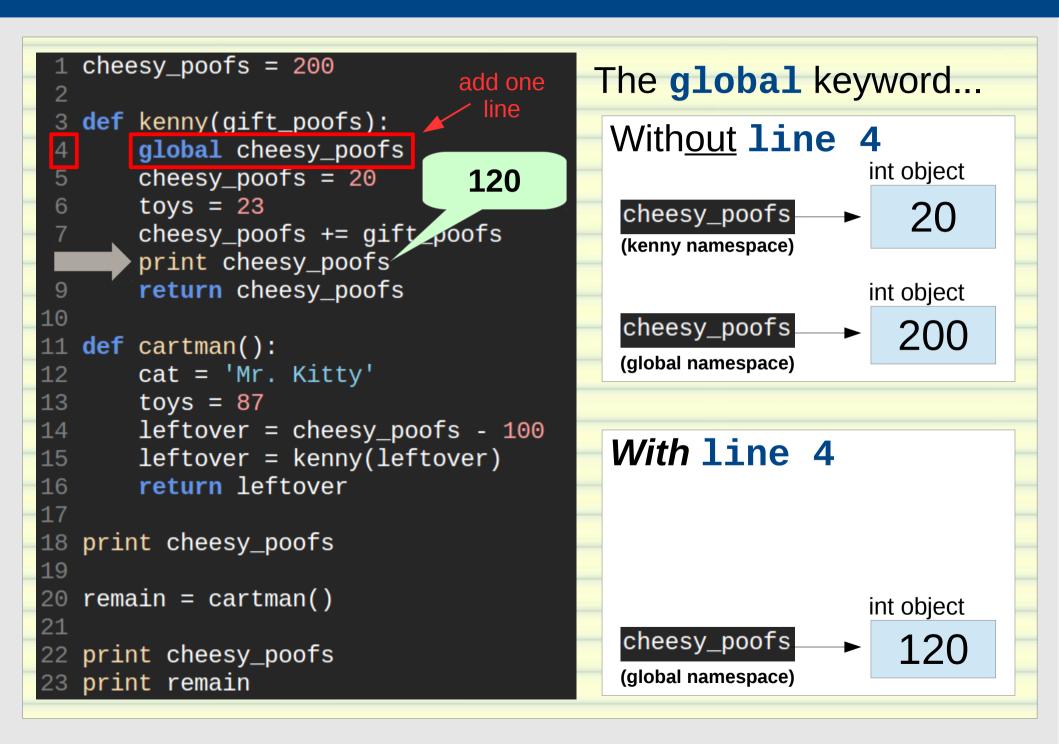
```
cheesy_poofs = 200
                                          The global keyword...
                               add one
                                 line
   def kenny(gift_poofs):
                                           Without line 4
       global cheesy_poofs
                                                               int object
       cheesy_poofs = 20
 6
       toys = 23
                                            cheesy_poofs
                                                                  20
       cheesy_poofs += gift_poofs
                                            (kenny namespace)
       print cheesy_poofs
       return cheesy_poofs
                                                               int object
10
                                            cheesy_poofs
                                                                 200
11 def cartman():
                                            (global namespace)
       cat = 'Mr. Kitty'
12
13
       toys = 87
       leftover = cheesy_poofs - 100
14
                                           With line 4
       leftover = kenny(leftover)
15
       return leftover
                                                               int object
16
17
                                                                  20
   print cheesy_poofs
19
   remain = cartman()
                                                               int object
21
                                            cheesy_poofs
                                                                 200
22 print cheesy_poofs
                                            (global namespace)
   print remain
```

```
cheesy_poofs = 200
                                          The global keyword...
                                add one
                                  line
   def kenny(gift_poofs):
                                           Without line 4
 4
5
       global cheesy_poofs
                                                               int object
       cheesy_poofs = 20
       toys = 23
                                            cheesy_poofs
                                                                  20
       cheesy_poofs += gift_poofs
                                            (kenny namespace)
       print cheesy_poofs
 8
       return cheesy_poofs
                                                               int object
10
                                            cheesy_poofs
                                                                 200
11 def cartman():
                                            (global namespace)
       cat = 'Mr. Kitty'
12
13
       toys = 87
       leftover = cheesy_poofs - 100
14
                                           With line 4
       leftover = kenny(leftover)
15
       return leftover
                                                               int object
16
17
                                                                  20
   print cheesy_poofs
19
   remain = cartman()
21
                                            cheesy_poofs
22 print cheesy_poofs
                                            (global namespace)
   print remain
```

Namespaces & Variable Scope

```
100
   cheesy_poofs = 200
                                          The global keyword...
                                add one
                                  line
   def kenny(gift_poofs):
                                           Without line 4
 4
5
       global cheesy_poofs
                                                               int object
       cheesy_poofs = 20
       toys = 23
                                            cheesy_poofs
                                                                  20
       cheesy_poofs += gift_poofs
                                            (kenny namespace)
       print cheesy_poofs
       return cheesy_poofs
                                                               int object
10
                                            cheesy_poofs
                                                                 200
11 def cartman():
                                            (global namespace)
       cat = 'Mr. Kitty'
12
13
       toys = 87
       leftover = cheesy_poofs - 100
14
                                           With line 4
       leftover = kenny(leftover)
15
       return leftover
                                                               int object
16
17
                                                                  20
   print cheesy_poofs
19
   remain = cartman()
                                                               int object
21
                                            cheesy_poofs
                                                                 120
22 print cheesy_poofs
                                            (global namespace)
   print remain
```

Namespaces & Variable Scope



Namespaces & Variable Scope

```
cheesy_poofs = 200
                               add one
                                 line
   def kenny(gift_poofs):
 4
5
       global cheesy_poofs
       cheesy_poofs = 20
       toys = 23
       cheesy_poofs += gift_poofs
       print cheesy_poofs
       return cheesy_poofs
10
  def cartman():
       cat = 'Mr. Kitty'
12
13
       toys = 87
       leftover = cheesy poofs
14
                                  100
       leftover = kenny
                           200
15
       return leftover
16
   print cheesy_poofs
                            120
19
   remain = cartman()
21
   print cheesy_poofs
   print remain
```

The **global** keyword...

lets you manipulate a global name inside your function's namespace

Whoa, can't see the forest for the trees!

Let's look at a simple, practical function to implement.

range()

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
         Returns a list containing an arithmetic progression of integers.
  11
         range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
  12
         it specifies the increment (or decrement). For example, range(4)
  13
         returns [0, 1, 2, 3]. The end point is omitted! These are exactly
  14
  15
         the valid indices for a list of 4 elements.
         11 11 11
  16
  17
  18
         numbers = []
         while start < stop:
  19
             numbers.append(start)
  20
  21
             start += step
  22
  23
         return numbers
  24
  25 for item in my_range(0, 10):
         print item
  26
```

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
          A simple implementation of range()
          my_range(start, stop[, step]) -> list of integers
  10
          Returns a list contai
  11
                                                                  of integers.
                                        <u>Takes 3 parameters</u>
          range(i, j) returns [
  12
                                                                  step is given,
          it specifies the incr
  13
                                                                  mple, range(4)
  14
                                     step has a <u>default value</u>
          returns [0, 1, 2, 3].
                                                                  hese are exactly
  15
          the valid indices for
          11 11 11
  16
  17
  18
         numbers = []
          while start < stop:
  19
  20
              numbers.append(start)
  21
              start += step
  22
  23
          return numbers
  24
  25 for item in my_range(0, 10):
  26
          print item
```

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
          Returns a list contai
  11
                                                                 of integers.
          range(i, j) returns []
  12
                                                                step is given,
                                   Can be called with or without
          it specifies the incr
  13
                                                                mple, range(4)
                                    specifying step parameter
  14
          returns [0, 1, 2, 3].
                                                                hese are exactly
  15
          the valid indices for
          11 11 11
  16
  17
  18
         numbers = []
         while start < stop:
  19
  20
              numbers.append(start)
  21
              start += step
  22
  23
          return numbers
  24
  25 for item in my_range(0, 10):
          print item
  26
```

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
         Returns a list containing an arithmetic progression of integers.
  11
  12
         range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
         it specifies the increment (or decrement). For example, range(4)
  13
         returns [0, 1, 2, 3]. The end point is omitted! These are exactly
  14
  15
         the valid indices for a list of 4 elements.
         11 11 11
  16
  17
         numbers = []
  18
                                                  returns a list
         while start < stop:</pre>
  19
             numbers.append(start)
  20
  21
             start += step
  22
  23
         return numbers
  24
  25 for item in my_range(0, 10):
         print item
  26
```

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
         Returns a list containing an arithmetic progression of integers.
  11
         range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
  12
         it specifies the increment (or decrement). For example, range(4)
  13
         returns [0, 1, 2, 3]. The end point is omitted! These are exactly
  14
         the valid indices for a list of 4 elements.
  15
         11 11 11
  16
                                             my_project$ python example.py
  17
  18
         numbers = []
         while start < stop:
  19
  20
             numbers.append(start)
  21
             start += step
  22
  23
         return numbers
  25 for item in my_range(0, 10):
         print item
                                             my_project$
```

You can reuse useful functions you write in other Python programs using

import

```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
         Returns a list containing an arithmetic progression of integers.
  11
         range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
  12
         it specifies the increment (or decrement). For example, range(4)
  13
         returns [0, 1, 2, 3]. The end point is omitted! These are exactly
  14
  15
         the valid indices for a list of 4 elements.
         11 11 11
  16
  17
  18
         numbers = []
         while start < stop:
  19
  20
             numbers.append(start)
  21
             start += step
  22
  23
         return numbers
  24
  25 for item in my_range(0, 10):
         print item
  26
```

```
test.py (in the same directory as example.py)
  import example
 # Compute sum of all evens from 2 to 20, inclusive
 total = 0
 for item in example.my_range(2, 21, 2):
     total += item
8 print total
                              my_project$ ls
                              example.py test.py
                              my_project$
```

```
test.py (in the same directory as example.py)
  import example
  total = 0
 for item in example.my_range(2, 21, 2):
      total += item
8 print total
                            everything from example.py
                              gets loaded into its own
                                   namespace
```

```
test.py (in the same directory as example.py)
  import example
  total = 0
 for item in example.my_range(2, 21, 2):
      total += item
8 print total
                            we access my_range() from
                                within the example
                               namespace using this
                                     notation
```

```
test.py (in the same directory as example.py)
        example import my_range
  # Compute sum of all evens from 2 to 20, inclusive
  total = 0
  for item in my_range(2, 21, 2):
    total += item
8 print total
                                                               <u>OR</u>
                                                    we can import my_range
                                                    into the global namespace
```

test.py (in the same directory as example.py)

```
from example import my_range as flower

from import my_range as flower

from example import my_range as flower

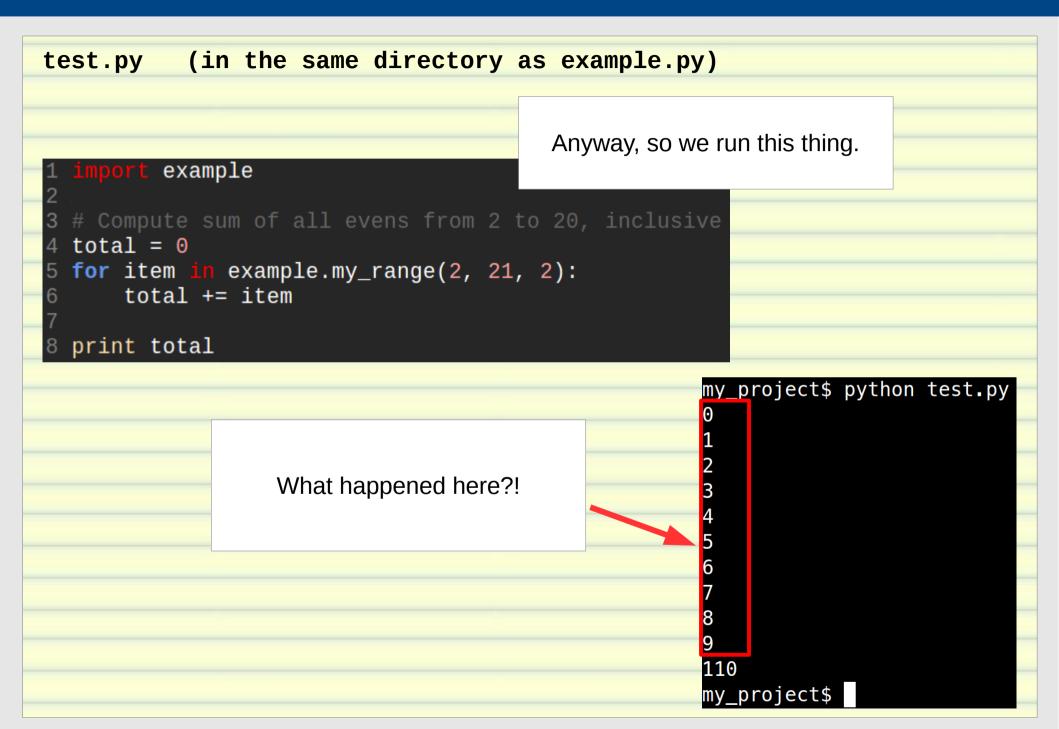
from import my_range as flower

f
```

if **my_range** is already being used in the global namespace

or if we just don't like the name my_range

we can import it into the global namespace using a custom name



```
example.py
   2 Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
         A simple implementation of range()
         my_range(start, stop[, step]) -> list of integers
  10
         Returns a list containing an arithmetic progression of integers.
  11
         range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
  12
         it specifies the increment (or decrement). For example, range(4)
  13
  14
         returns [0, 1, 2, 3]. The
                                                                   are exactly
  15
         the valid indices for a li
                                         When Python imported
         11 11 11
  16
                                      example.py, everything was
  17
                                               executed.
         numbers = []
  18
         while start < stop:
  19
                                           INCLUDING THIS!!
             numbers.append(start)
  20
  21
             start += step
  22
  23
         return numbers
  25 for item in my_range(0, 10):
         print item
```

```
example.py
     Simple, demonstrative example of range()
     def my_range(start, stop, step=1):
          A simple implementation of range()
         my_range(start, stop[, ste
  10
                                             What do we do?
  11
          Returns a list containing
                                                                      ntegers.
          range(i, j) returns [i, i+
  12
                                                                      is given,
          it specifies the increment
  13
                                                 Delete it?
                                                                       range(4)
  14
          returns [0, 1, 2, 3]. The
                                                                      are exactly
          the valid indices for a li
  15
                                         no... it's nice to be able to
          11 11 11
  16
                                         test functions in the same
  17
                                          file you wrote them in...
  18
         numbers = []
         while start < stop:
  19
  20
              numbers.append(start)
              start += step
  22
  23
          return numbers
  25 for item in my_range(0, 10):
          print item
```

```
example.py
        2 Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
       11
              Returns a list containing an arithmetic progression of integers.
       12
              range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
              it specifies the increment (or decrement). For example, range(4)
       13
              returns [0, 1, 2, 3]. The end point is omitted! These are exactly
       14
              the valid indices for a list of 4 elements.
       15
               11 11 11
       16
       17
       18
              numbers = []
              while start < stop:
       19
                                                       Change it to this.
       20
                  numbers.append(start)
                  start += step
       23
              return numbers
       25 def main():
              for item in my_range(0, 10):
       26
                  print item
          if __name__ == "__main__":
              main()
```

```
example.py
        2 Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
              Returns a list containing an arithmetic progression of integers.
       11
       12
              range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
              it specifies the increment (or decrement). For example, range(4)
       13
              returns [0, 1, 2, 3]. The end point is omitted! These are exactly
       14
       15
              the valid indices for a list of 4 elements.
              0.00
       16
       17
       18
              numbers = []
              while start < stop:
       19
       20
                  numbers.append(start)
                  start += step
                                                   This looks confusing!
       23
              return numbers
       25 def main():
              for item in my_range(0, 10):
       26
                  print item
          if name == " main ":
              main()
```

```
example.py
        2 Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
              Returns a list containing an arithmetic progression of integers.
       11
       12
              range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
              it specifies the increment (or decrement). For example, range(4)
       13
              returns [0, 1, 2, 3]. The end point is omitted! These are exactly
       14
              the valid indices for a list of 4 elements.
       15
               11 11 11
       16
       17
       18
              numbers = []
              while start < stop:
       19
                  numbers.append(start)
       20
                                                         This is a
                   start += step
                                                     "magic variable"
       23
              return numbers
       25 def main():
       26
              for item in my_range(0, 10):
       27
                  print itca
       29 if
             __name__ == "__main__":
              ma⊥n()
```

```
example.py
          Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
       11
              Returns a list containing an aris
                                                                        agers.
                                                        This is a
       12
              range(i, j) returns [i, i+1, i+2]
                                                                          iven,
       13
              it specifies the increment (or de
                                                   "magic variable"
                                                                          ge(4)
              returns [0, 1, 2, 3]. The end po
       14
                                                                          exactly
       15
              the valid indices for a list of
                                                Python sets it value to
              11 11 11
       16
       18
              numbers = []
                                                         main "
       19
              while start < stop:
       20
                  numbers.append(start)
                                                when it is running a file
                  start += step
                                                from the command line:
       23
              return numbers
                                                $ python example.py
       25 def main():
              for item in my_range(0, 10):
       26
       27
                  print itca
             main()
```

```
example.py
          Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
                                                         This is a
                                                    "magic variable"
              A simple implementation of range
                                                 Python sets it value to
              my_range(start, stop[, step]) ->
       10
       11
              Returns a list containing an arid
                                                                           ers.
                                                          main "
       12
              range(i, j) returns [i, i+1, i+2]
                                                                           iven,
       13
              it specifies the increment (or de
                                                                           ge(4)
              returns [0, 1, 2, 3]. The end po
                                                 when it is running a file
       14
                                                                           exactly
       15
              the valid indices for a list of
                                                 from the command line:
              11 11 11
       16
                                                $ python example.py
       18
              numbers = []
              while start < stop:
       19
       20
                  numbers.append(start)
                                                  or when it is running
                  start += step
                                                   in interactive mode:
       23
              return numbers
                                                >>> print __name___
       25 def main():
                                                  main
       26
              for item in my_range(0, 10):
       27
                  print itcm
          if
             __name__ == "__main__":
              main()
```

```
example.py
          Simple, demonstrative example of rand
        5 def my_range(start, stop, step=1):
              A simple implementation of range
              my_range(start, stop[, step]) ->
        10
       11
              Returns a list containing an arid
       12
              range(i, j) returns [i, i+1, i+2,
       13
              it specifies the increment (or de
       14
              returns [0, 1, 2, 3]. The end po
       15
              the valid indices for a list of
       16
       17
              numbers = []
       18
       19
              while start < stop:
       20
                   numbers.append(start)
                   start += step
       23
              return numbers
       25 def main():
       26
              for item in my range(0, 10):
       27
                   print itcm
          if __name__ == "__main__":
              main()
```

This is a "magic variable"

Python sets it value to

"___main___"

when it is running a file from the command line:

\$ python example.py

or when it is running in interactive mode:

>>> print __name__
main

<u>BUT</u>

__name__

is NOT set when a file is loaded using

import

ers. liven, lge(4) exactly

```
example.py
        2 Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
              Returns a list containing an arithmetic progression of integers.
       11
       12
              range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
              it specifies the increment (or decrement). For example, range(4)
       13
              returns [0, 1, 2, 3]. The end po
       14
                                                                            exactlv
       15
              the valid indices for a list of
               11 11 11
       16
                                                       So, when you
       17
                                                      import this file
       18
              numbers = []
              while start < stop:
       19
       20
                  numbers.append(start)
                                                          main()
                  start += step
                                                   will not be executed
       23
              return numbers
       25 def main():
              for item in my_range(0, 10):
       26
       27
                  print itca
       29 if __name__ == "__main__":
              main()
```

```
example.py
        2 Simple, demonstrative example of range()
        5 def my_range(start, stop, step=1):
              A simple implementation of range()
              my_range(start, stop[, step]) -> list of integers
       10
              Returns a list containing an arithmetic progression of integers.
       11
       12
              range(i, j) returns [i, i+1, i+2, ..., j-1]. When step is given,
              it specifies the increment (or decrement). For example, range(4)
       13
              returns [0, 1, 2, 3]. The end po
       14
                                                                            exactlv
       15
              the valid indices for a list of
               11 11 11
       16
                                                     Files that can be
       17
       18
              numbers = []
                                                        imported
              while start < stop:
       19
       20
                  numbers.append(start)
                  start += step
                                                      are commonly
                                                      called Modules
       23
              return numbers
       25 def main():
       26
              for item in my_range(0, 10):
       27
                  print itca
       29 if __name__ == "__main__":
              main()
```

Questions?