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

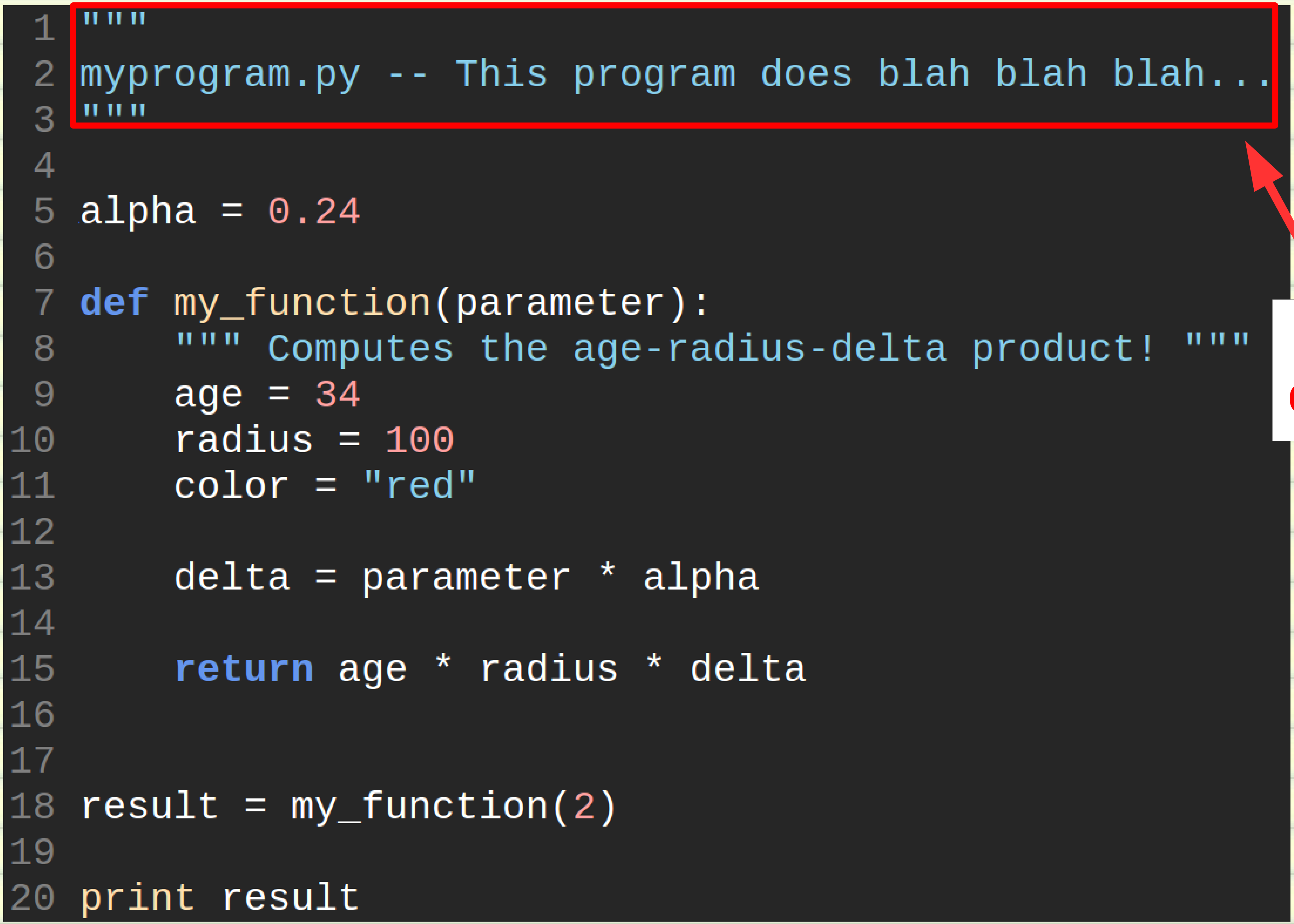
Textbook

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

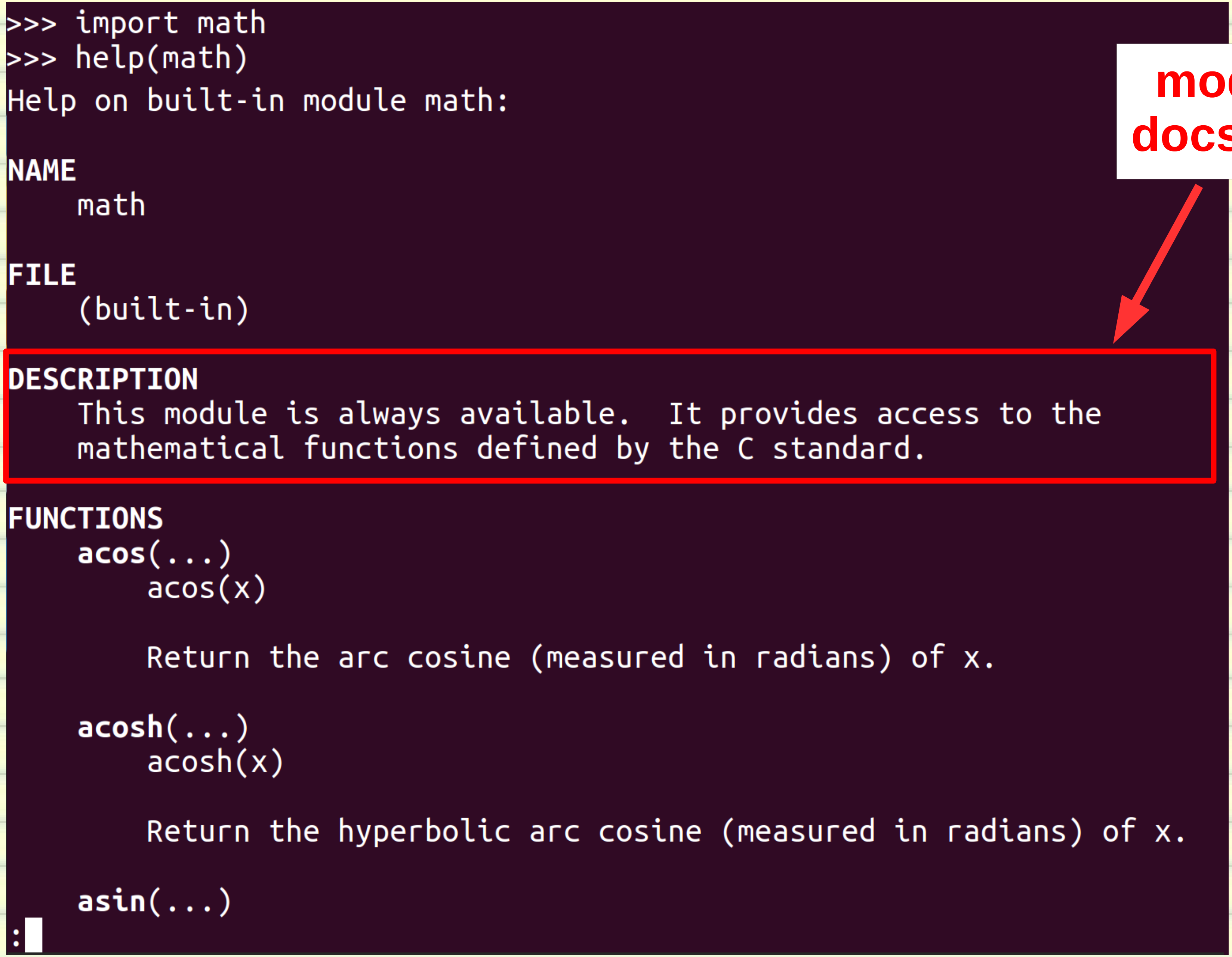
Anatomy of an (almost) “proper” Python program



```
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

module docstring

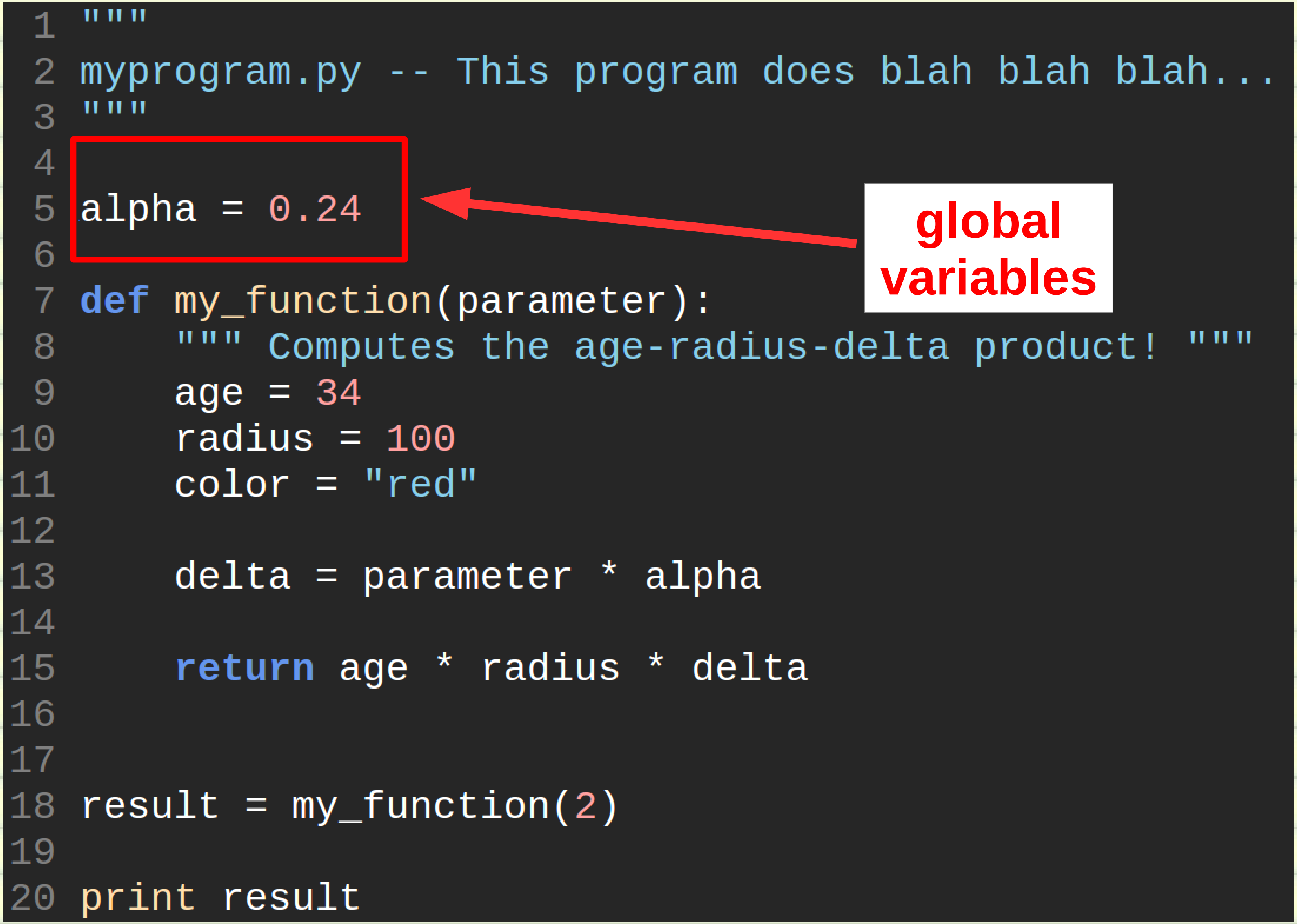
Anatomy of an (almost) “proper” Python program



```
>>> import math
>>> help(math)
Help on built-in module math:

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(...)
:
```

module docstring

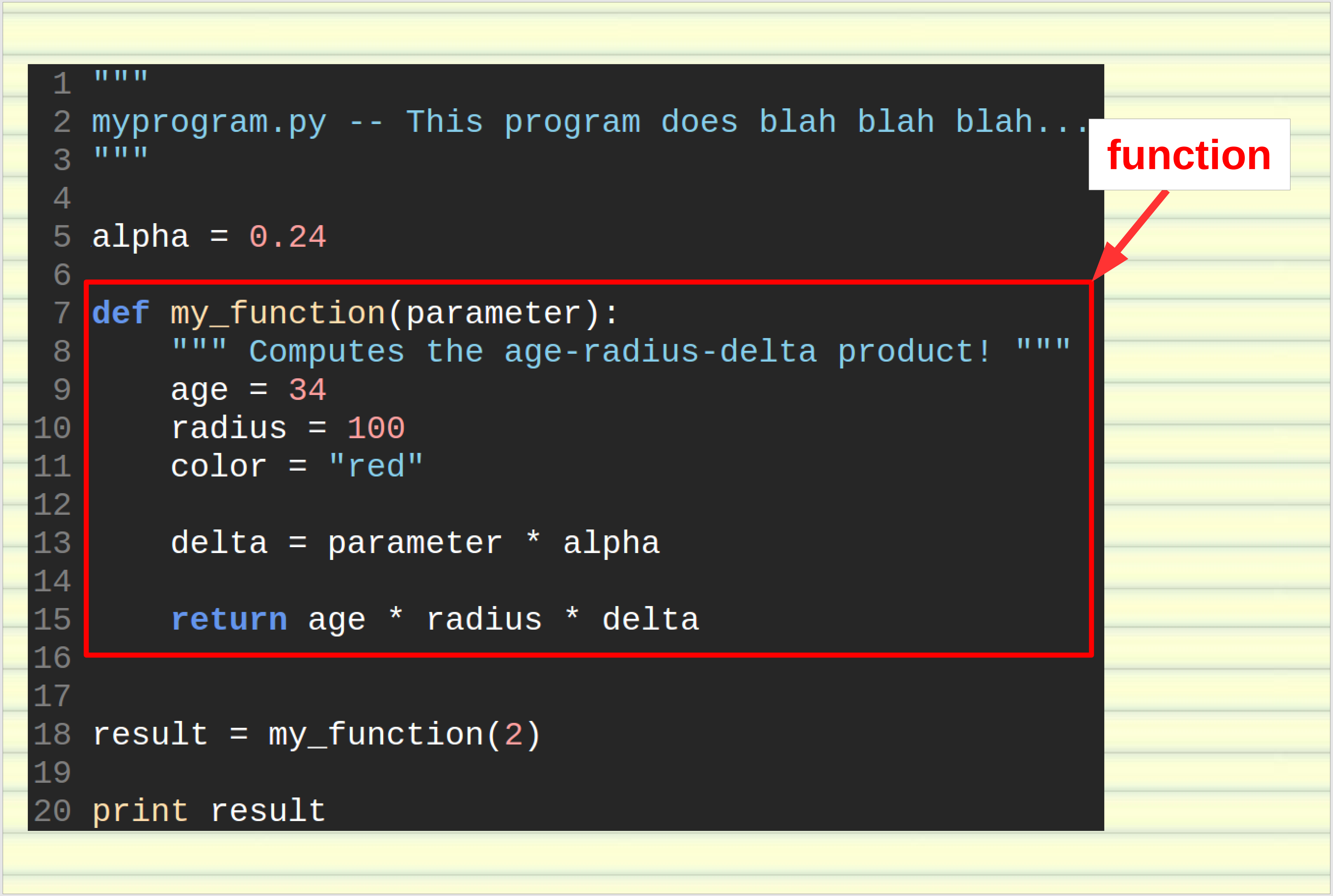
Anatomy of an (almost) “proper” Python program



```
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

global variables

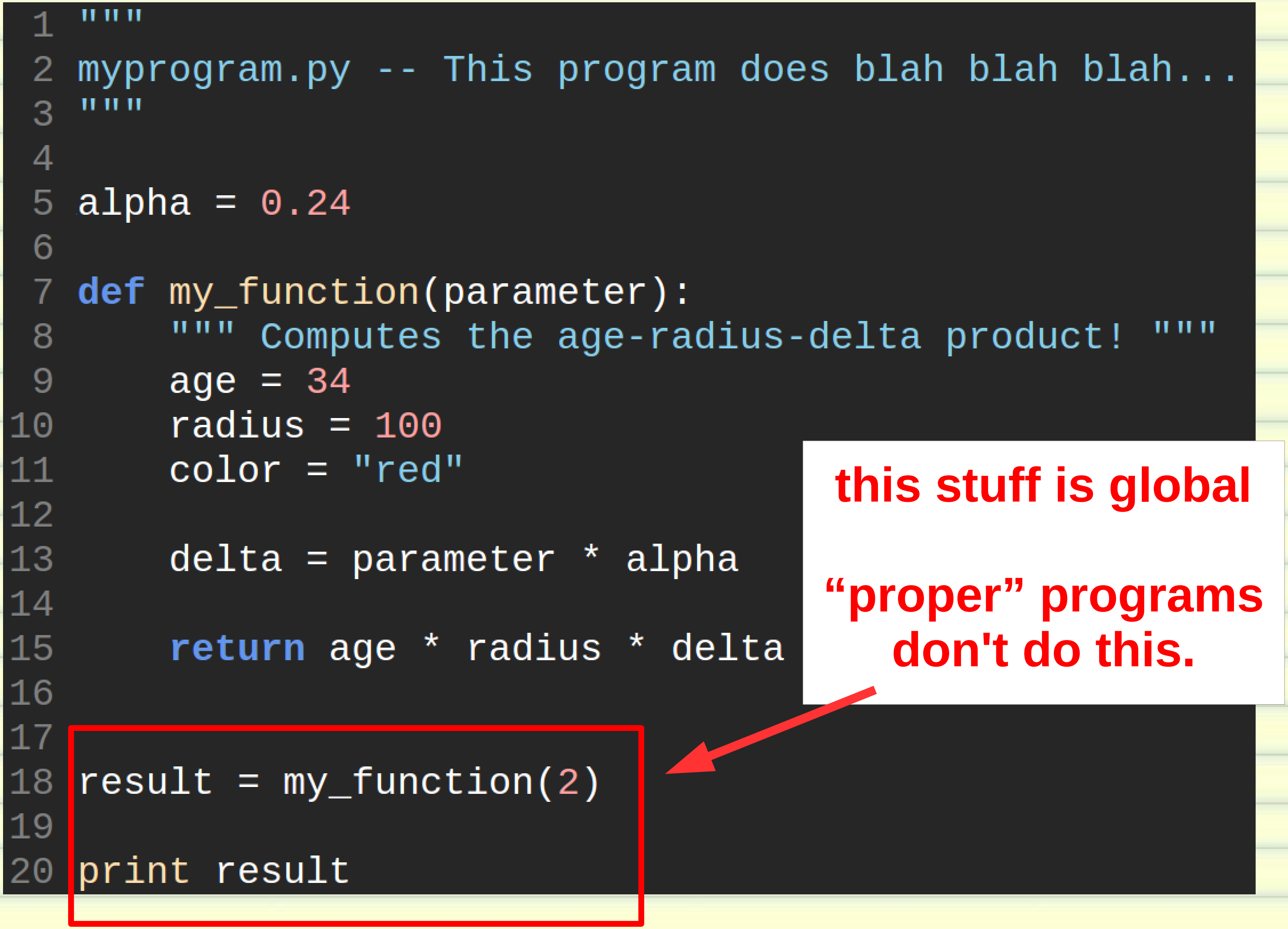
Anatomy of an (almost) “proper” Python program



```python
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

function

Anatomy of an (almost) “proper” Python program

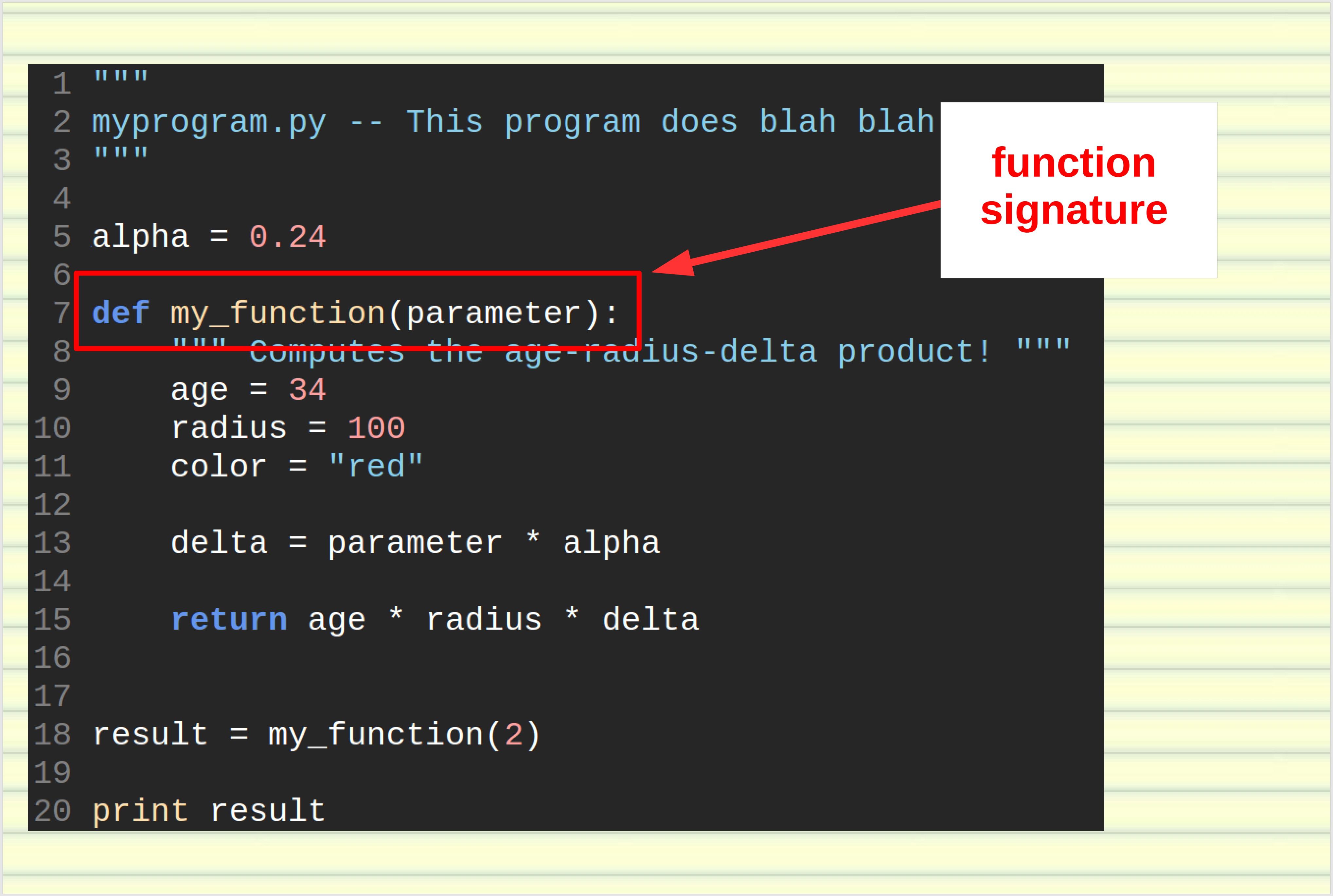


```python
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

this stuff is global

“proper” programs don't do this.

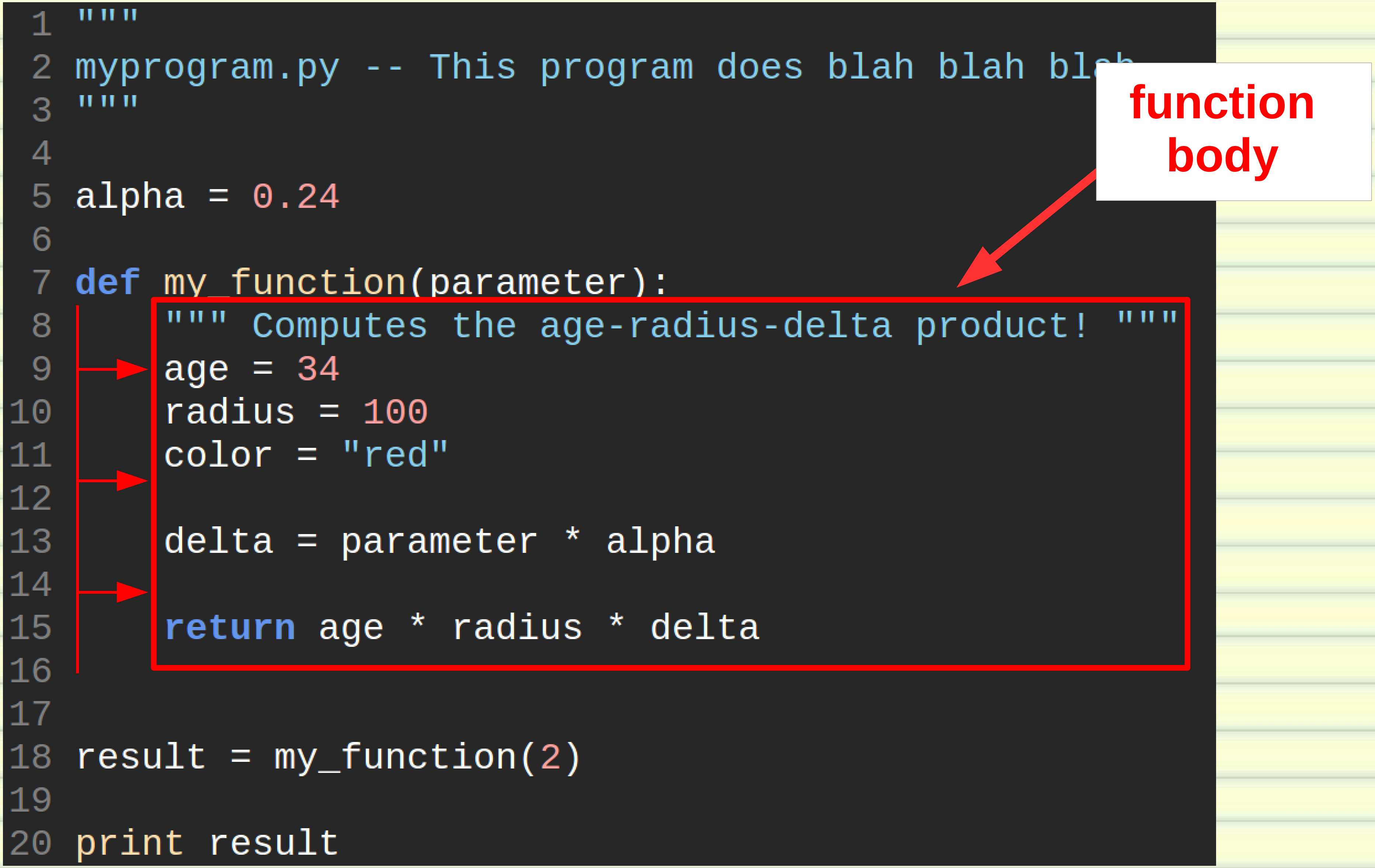
Anatomy of a Python function



```python
"""
myprogram.py -- This program does blah blah
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

function signature

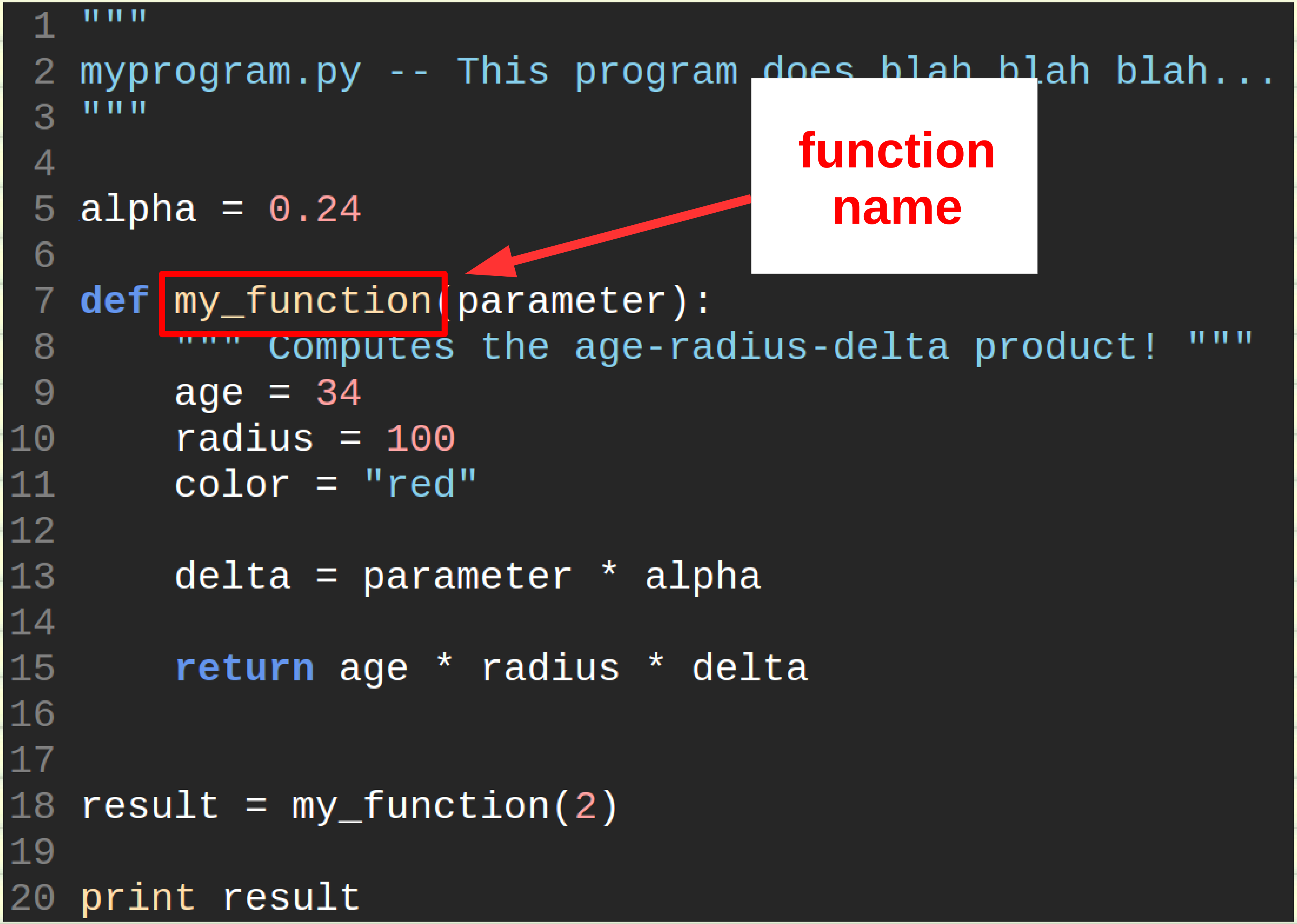
Anatomy of a Python function



```python
"""
myprogram.py -- This program does blah blah blah
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

function body

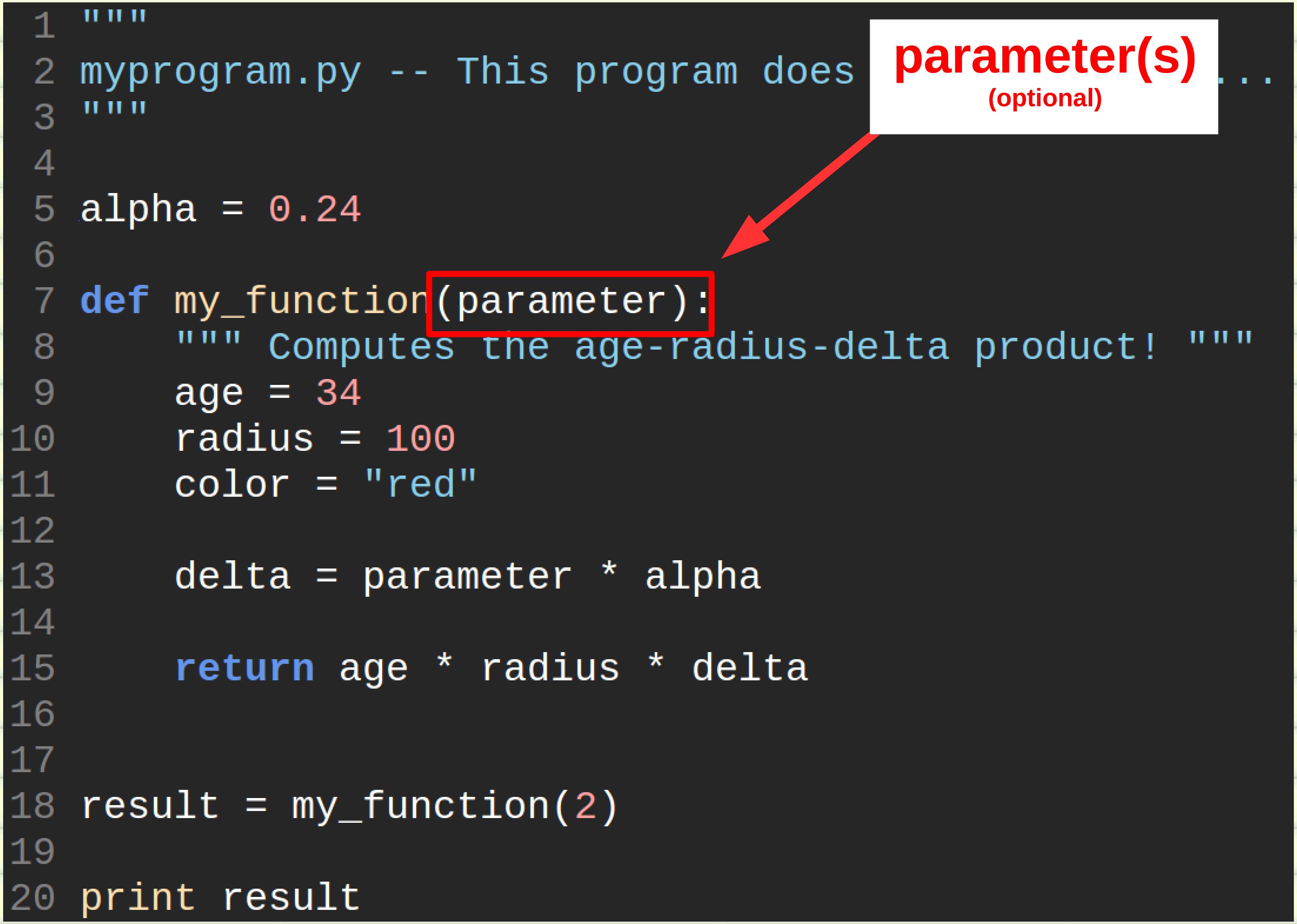
Anatomy of a Python function



```
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

function name

Anatomy of a Python function

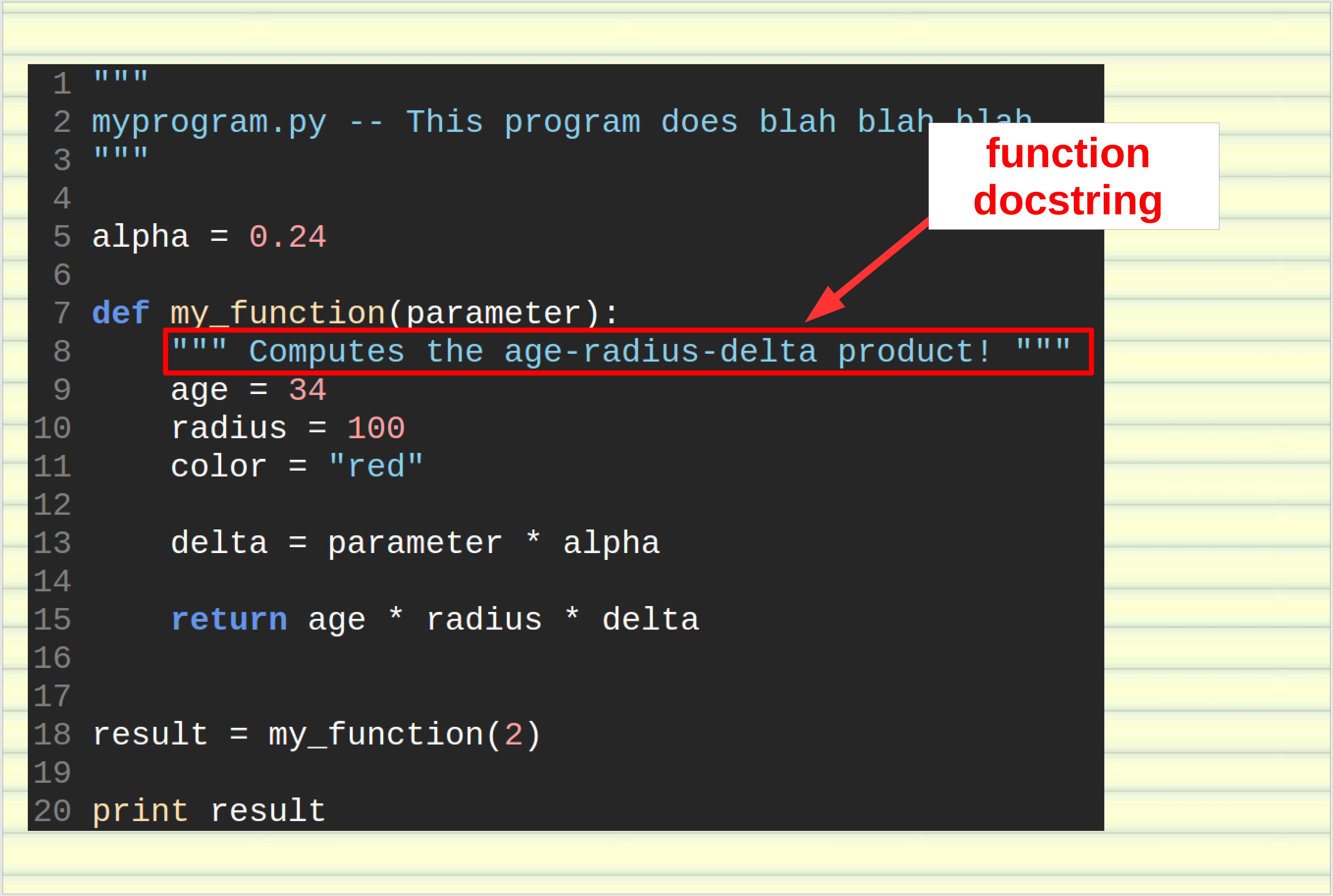


```
"""
myprogram.py -- This program does ...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

parameter(s)

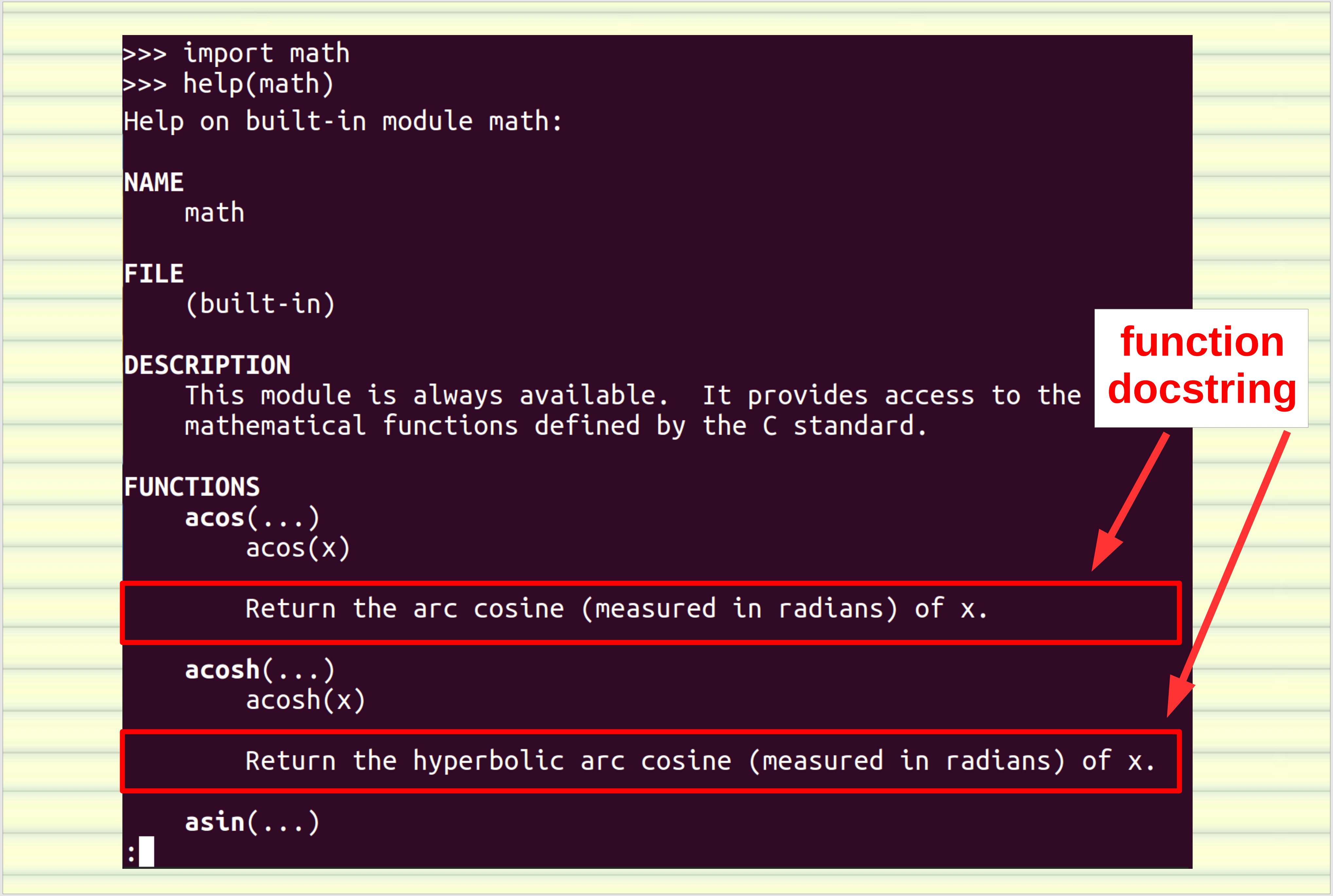
(optional)

Anatomy of a Python function



```
"""
myprogram.py -- This program does blah blah blah
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

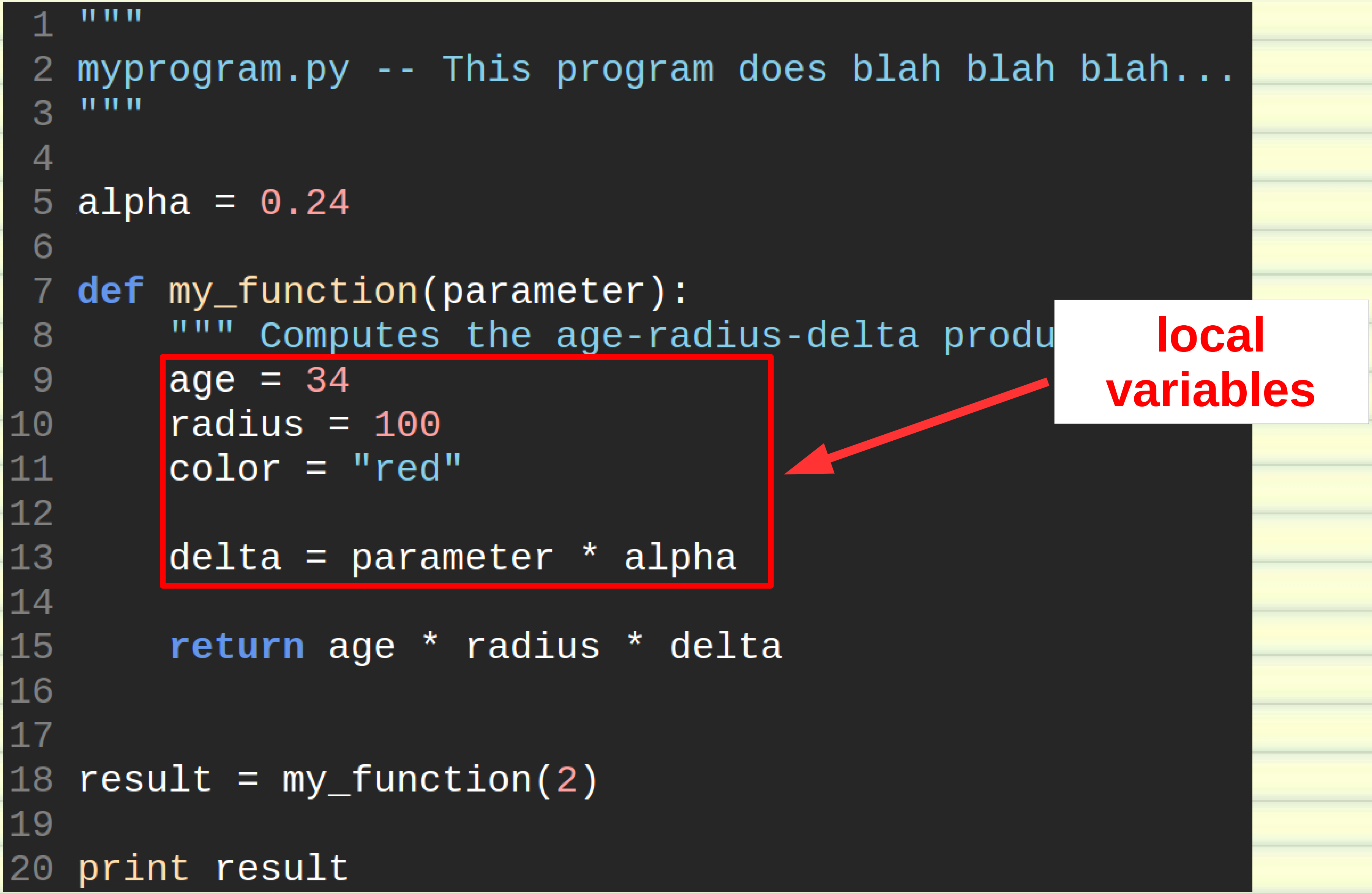
Anatomy of an (almost) “proper” Python program



```
>>> import math
>>> help(math)
Help on built-in module math:

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(...)
:
```

function docstring

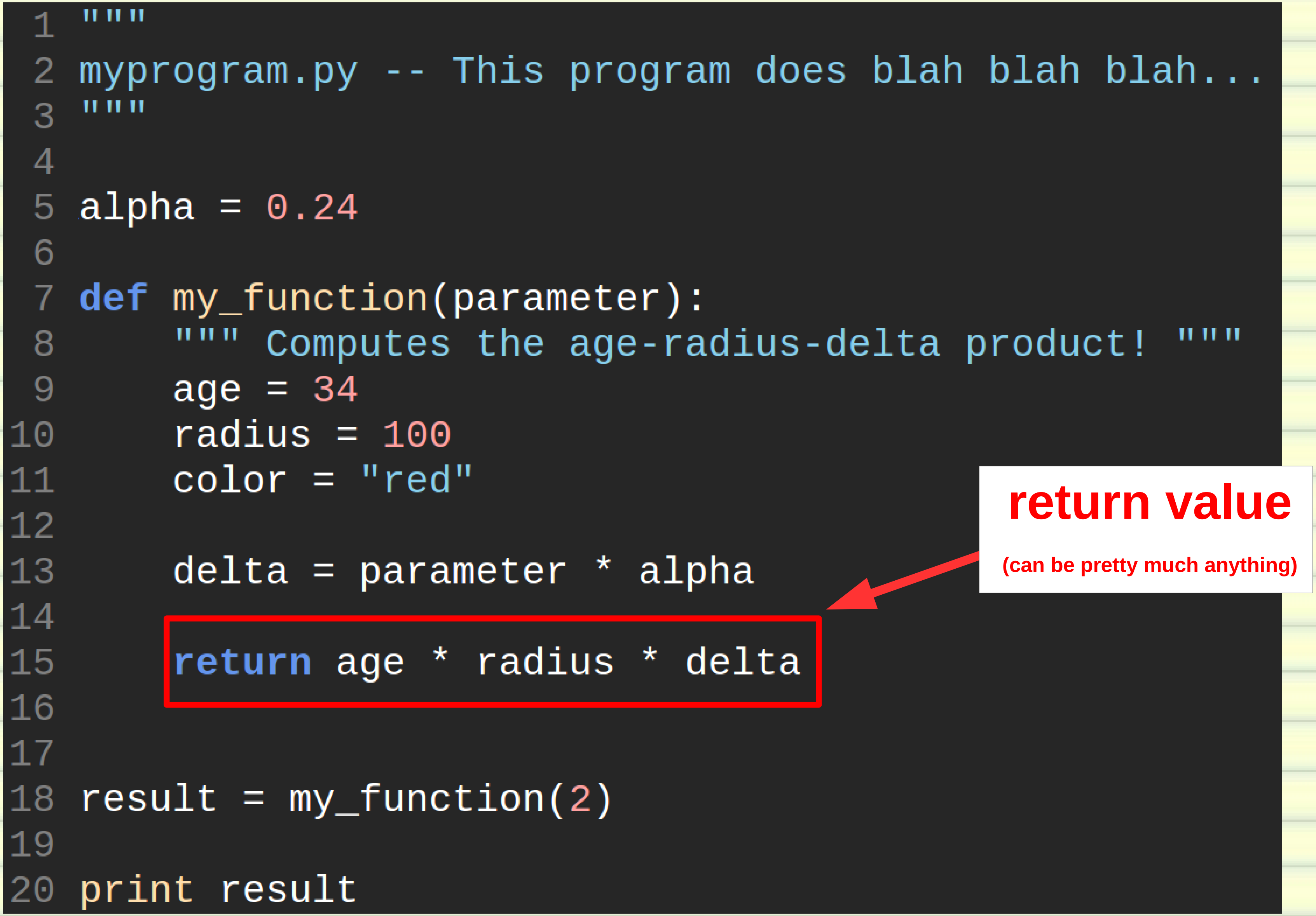
Anatomy of a Python function



```
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta produ
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

local variables

Anatomy of a Python function

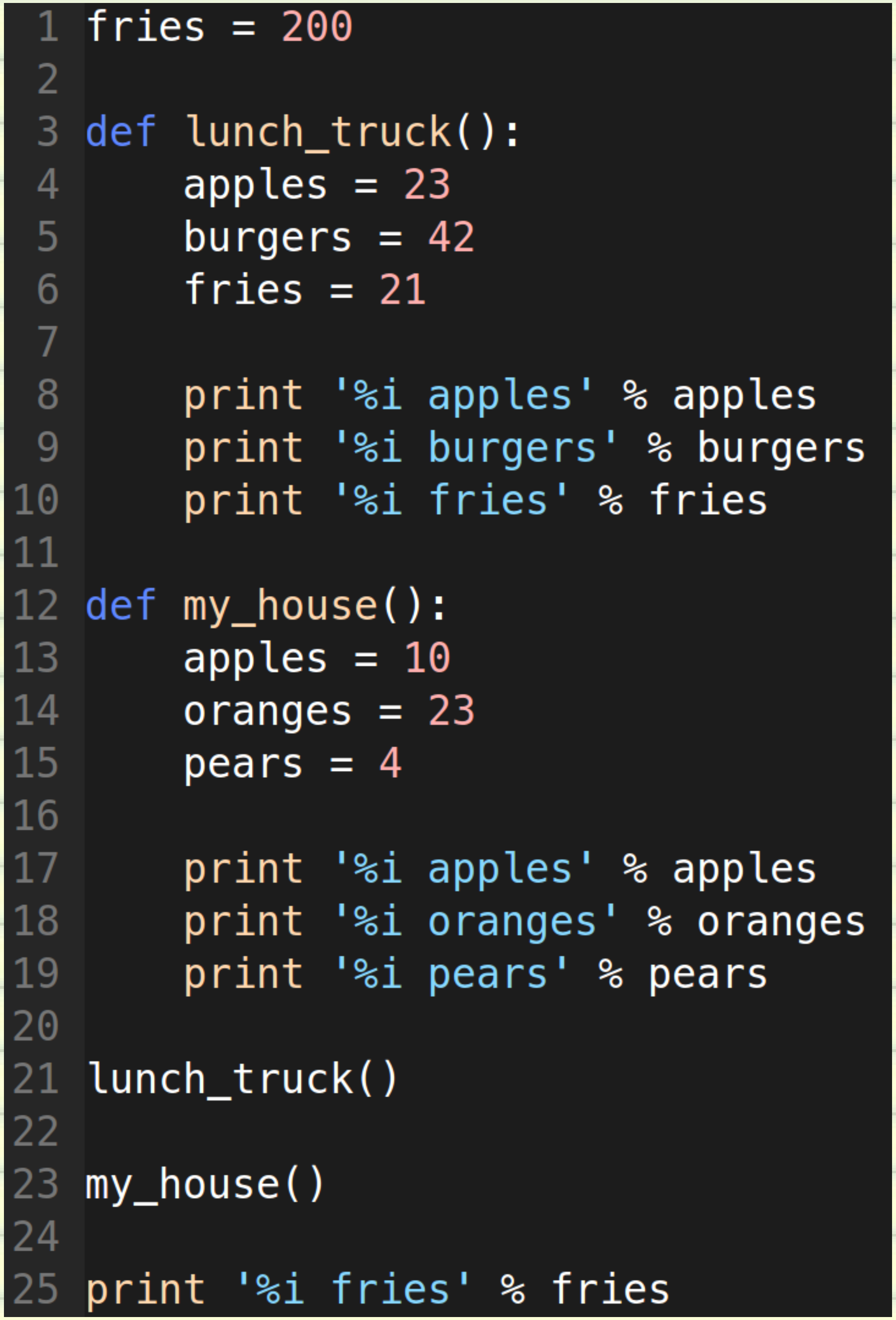


```
"""
myprogram.py -- This program does blah blah blah...
"""

alpha = 0.24

def my_function(parameter):
    """ Computes the age-radius-delta product! """
    age = 34
    radius = 100
    color = "red"

    delta = parameter * alpha

    return age * radius * delta


result = my_function(2)

print result
```

return value

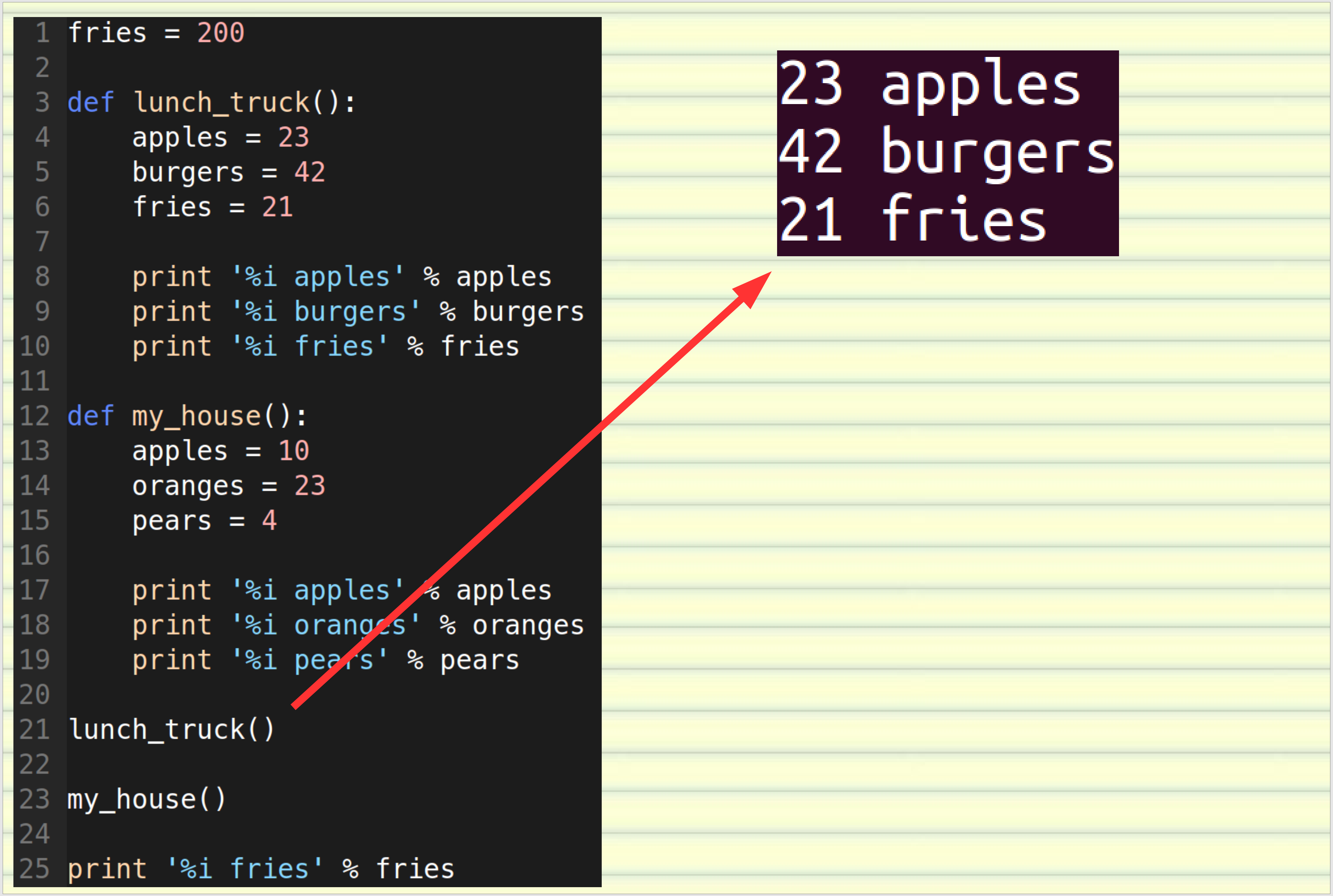
(can be pretty much anything)

Namespaces & Variable Scope



```python
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

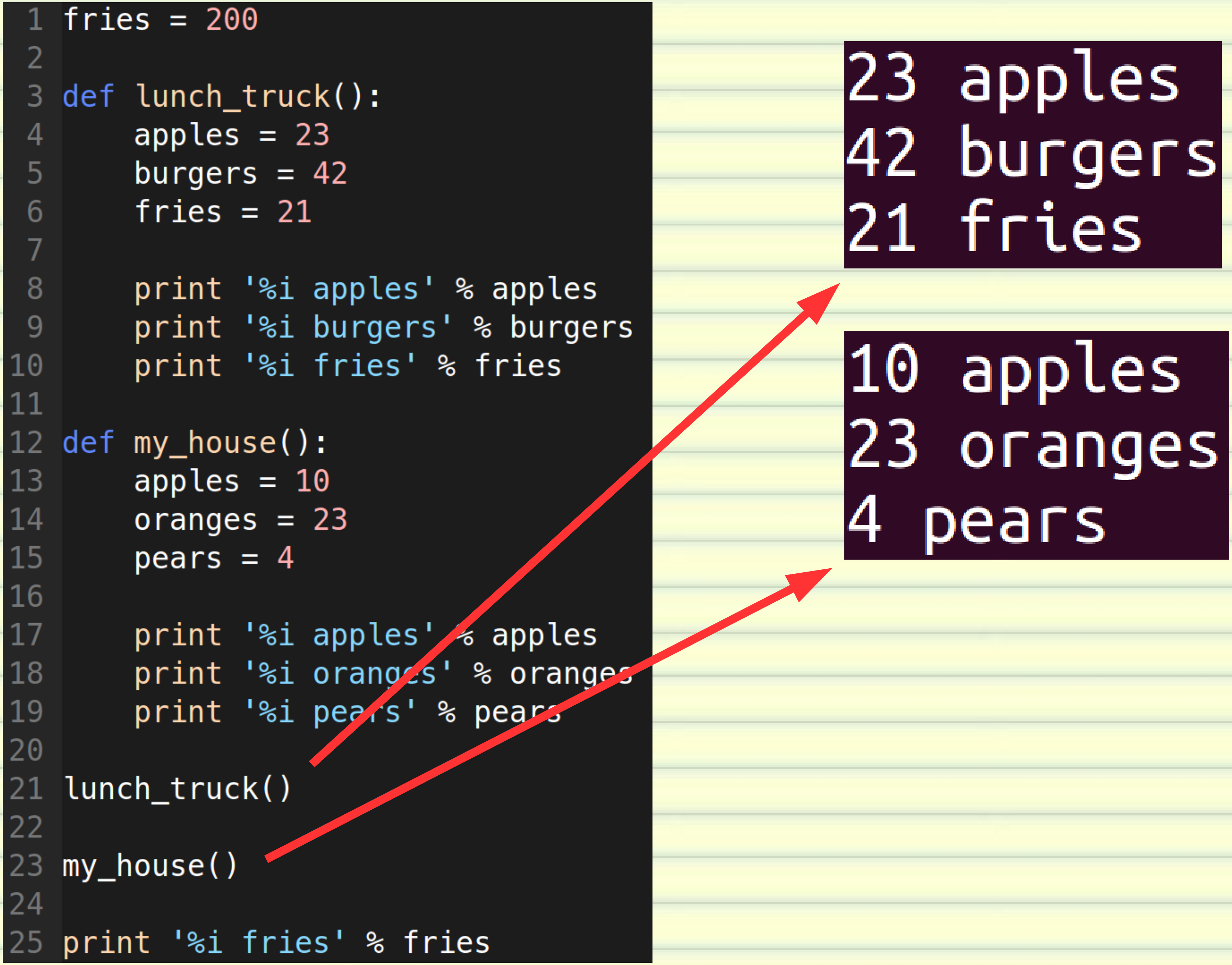
Namespaces & Variable Scope



```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

```
23 apples
42 burgers
21 fries
```

Namespaces & Variable Scope

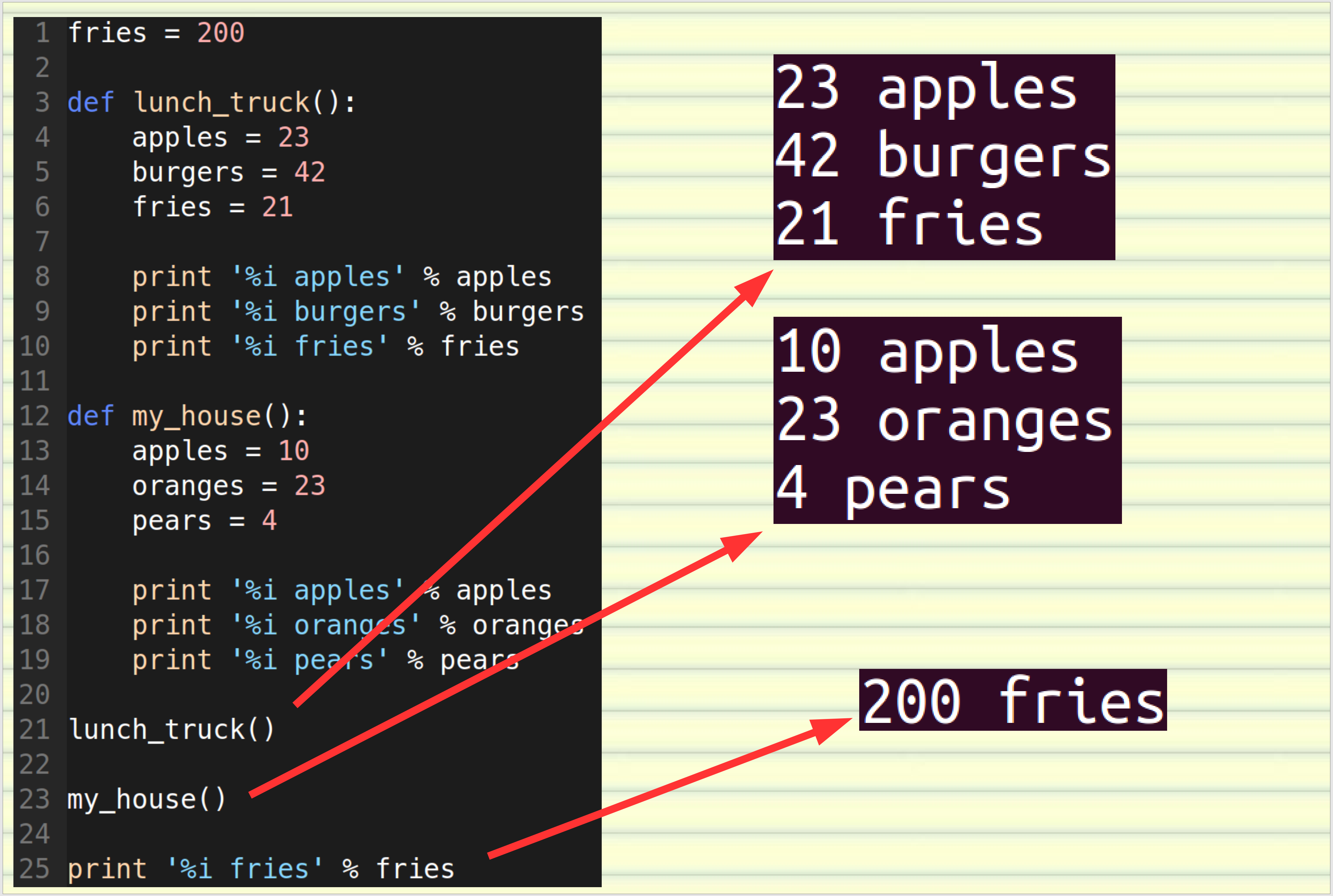


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

```
23 apples
42 burgers
21 fries
```

```
10 apples
23 oranges
4 pears
```

Namespaces & Variable Scope



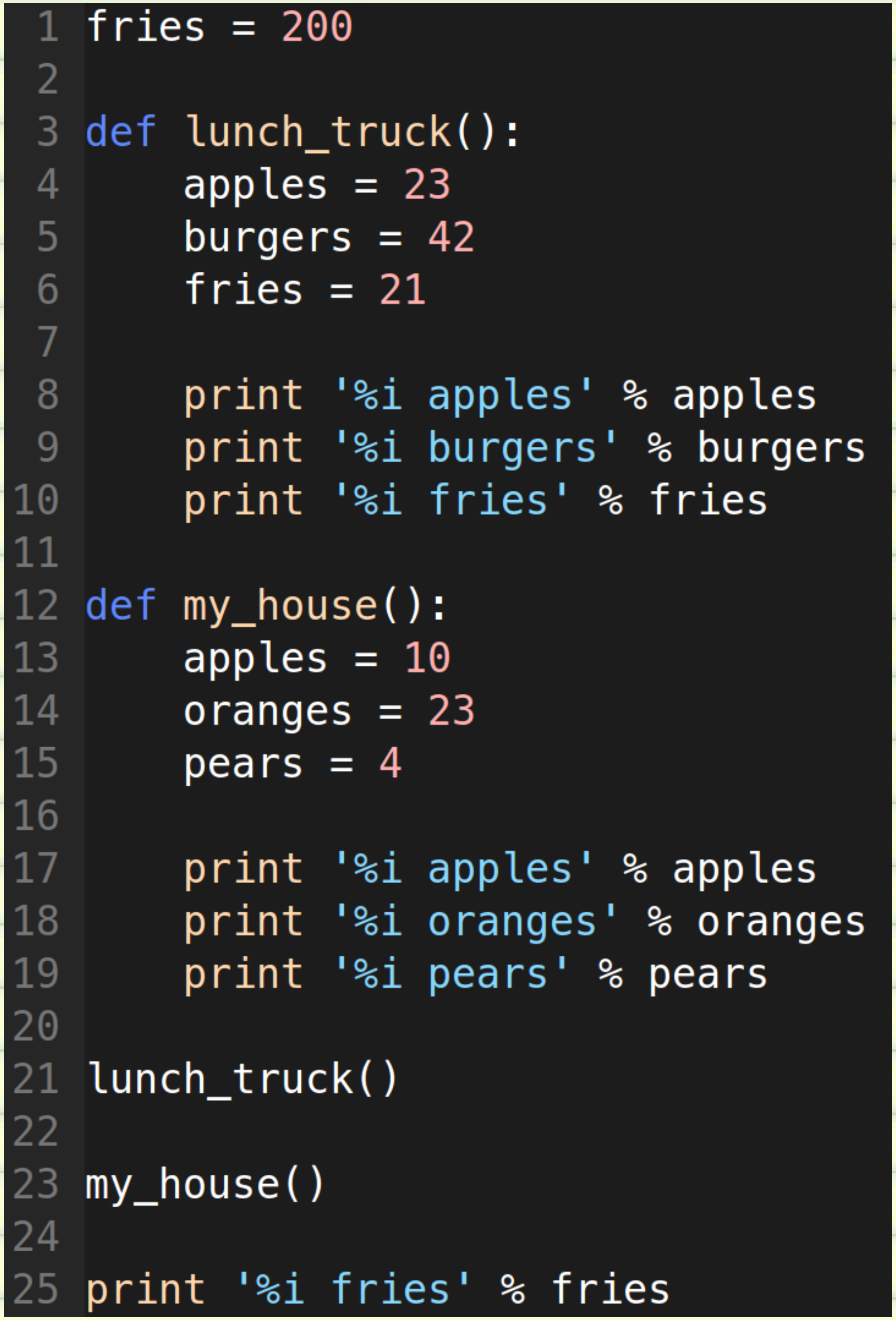
```python
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

```
23 apples
42 burgers
21 fries
```

```
10 apples
23 oranges
4 pears
```

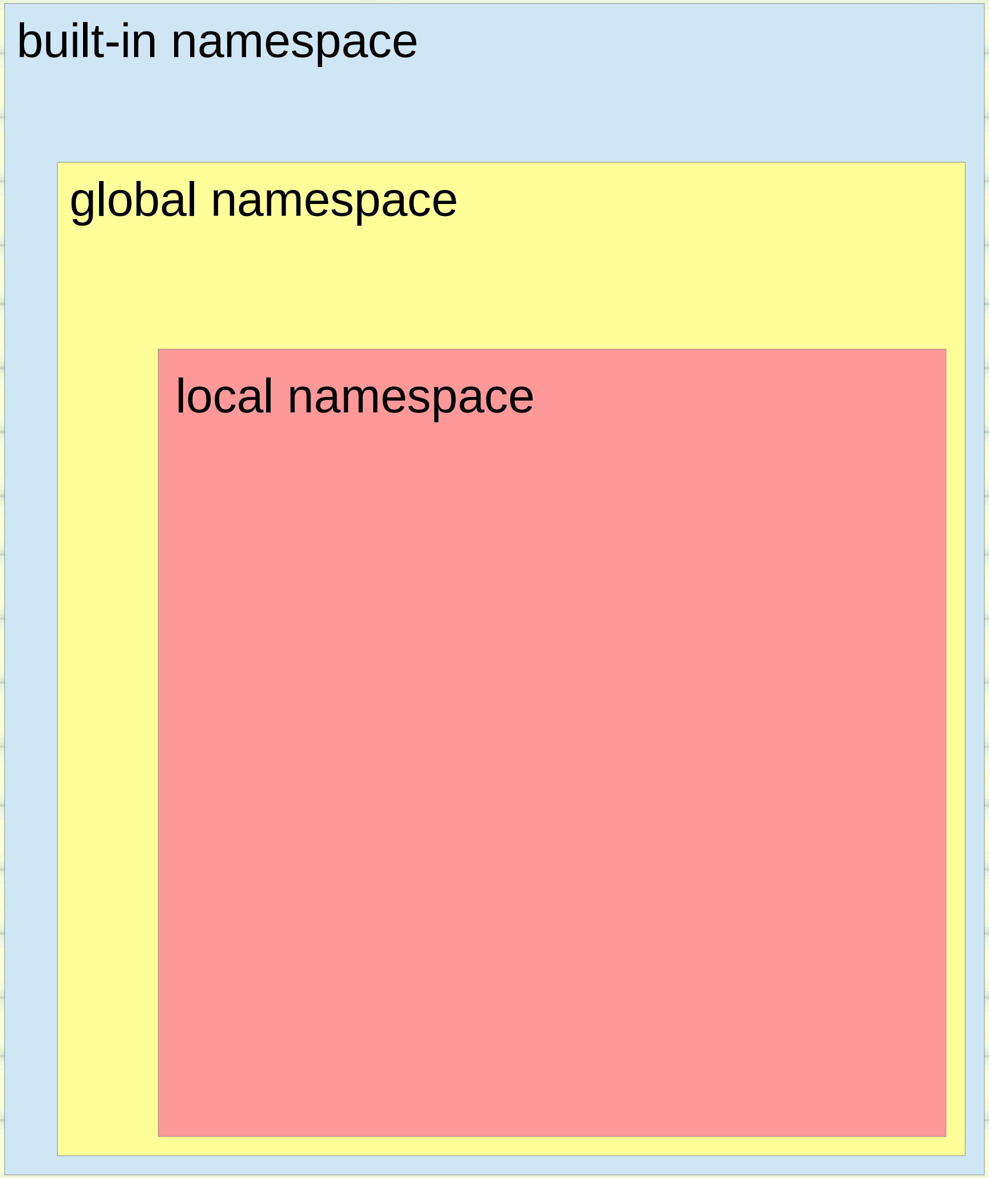
```
200 fries
```

Namespaces & Variable Scope

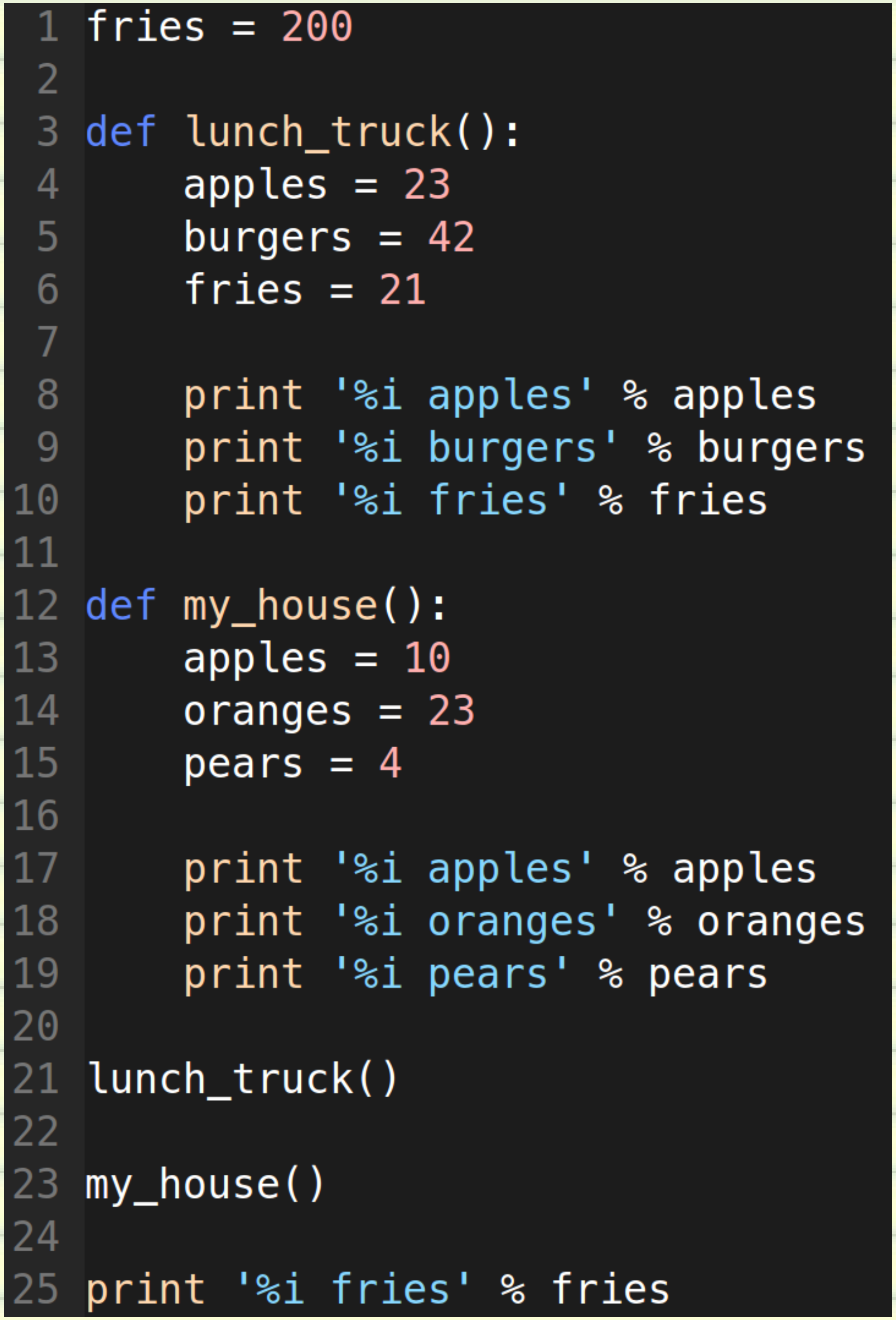


```python
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Name search looks like this:

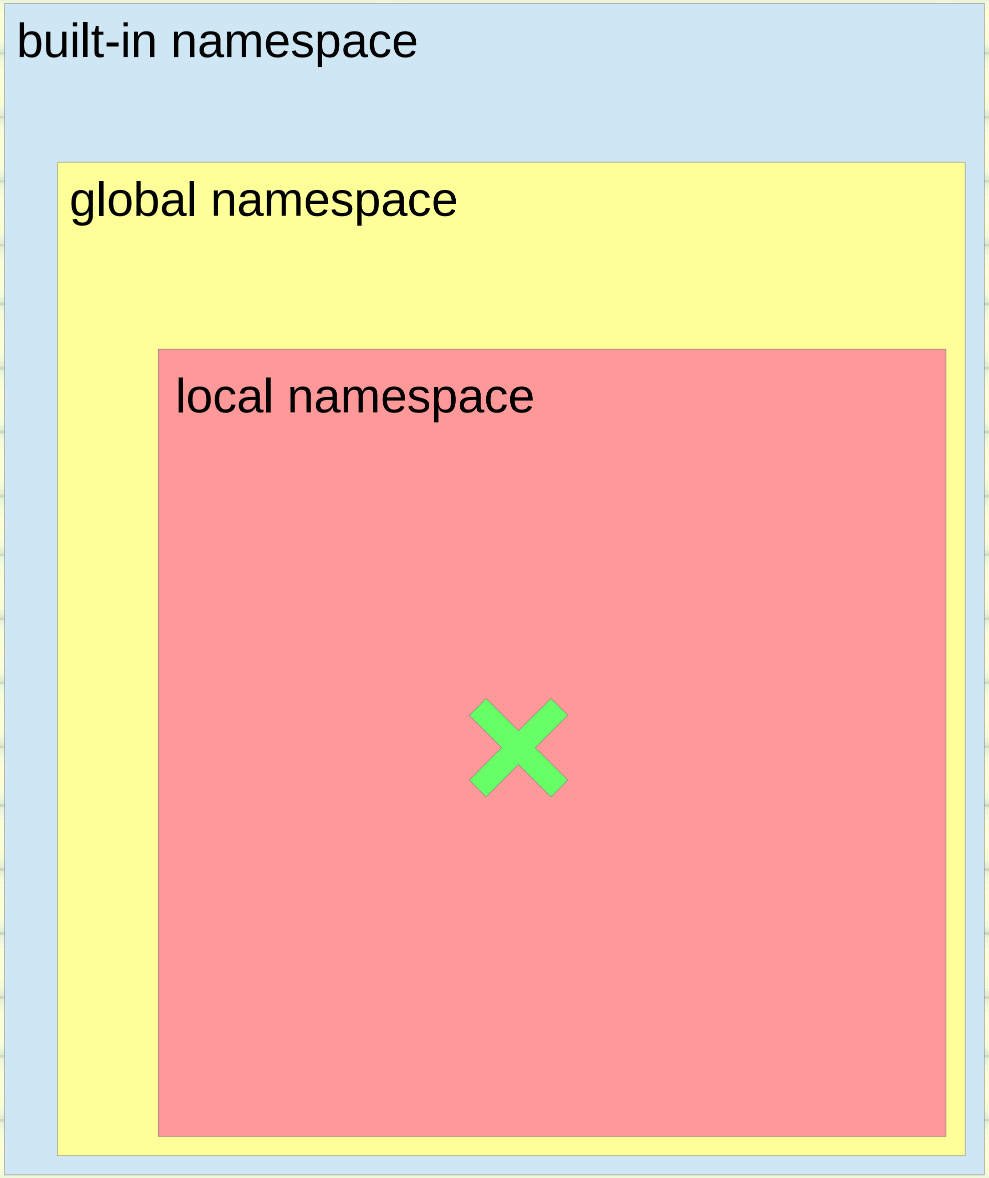


Namespaces & Variable Scope



```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Name search looks like this:

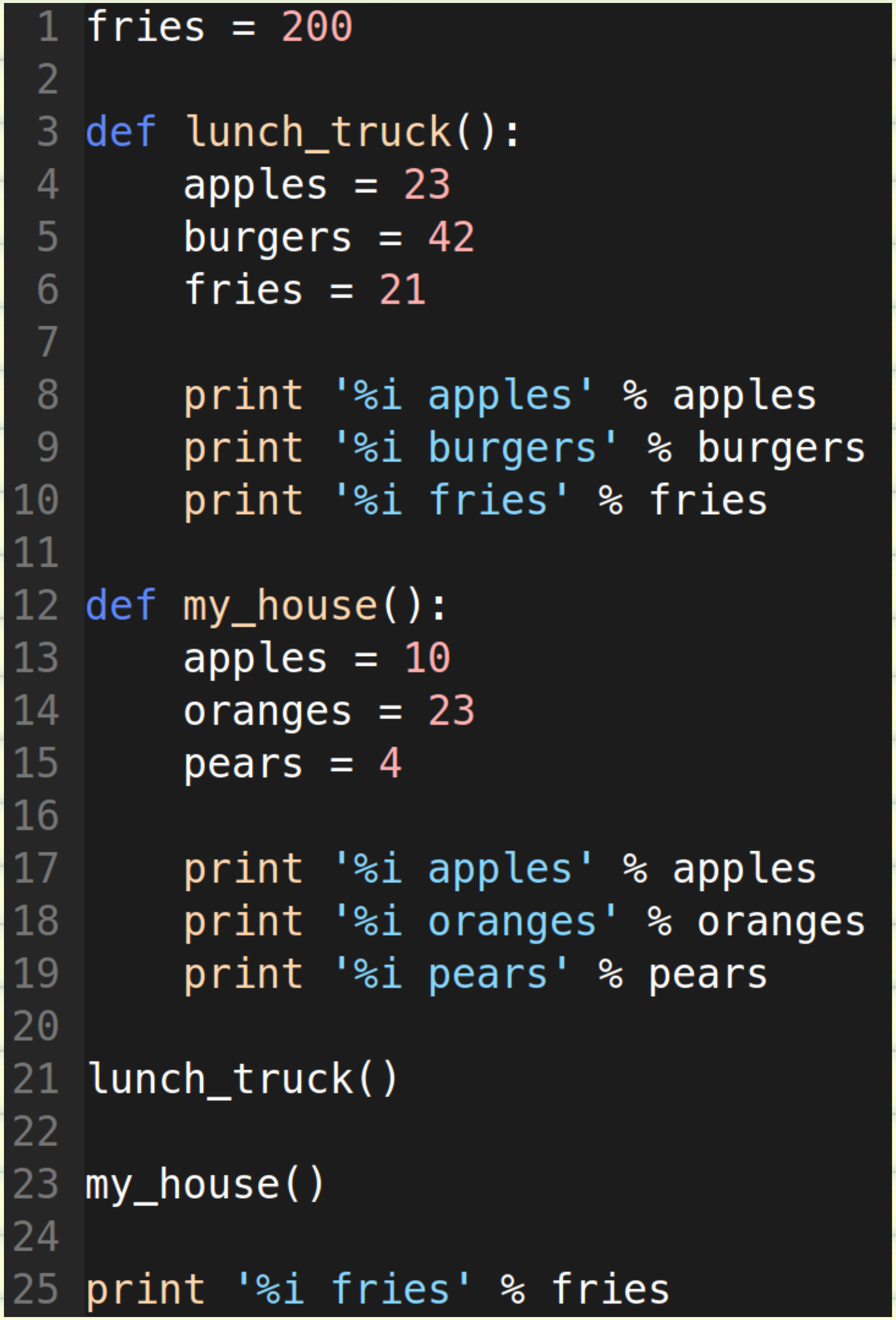


built-in namespace

global namespace

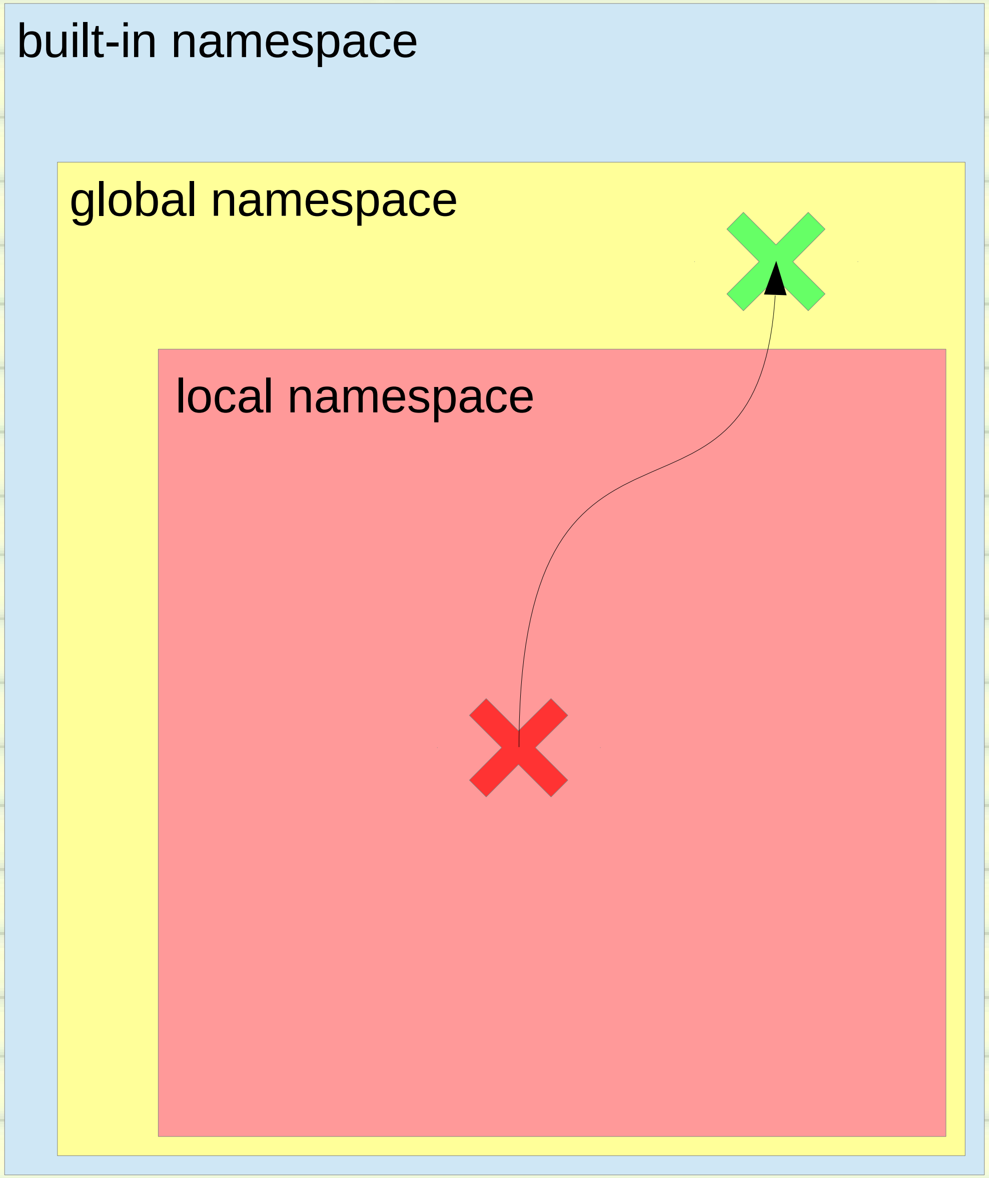
local namespace

Namespaces & Variable Scope

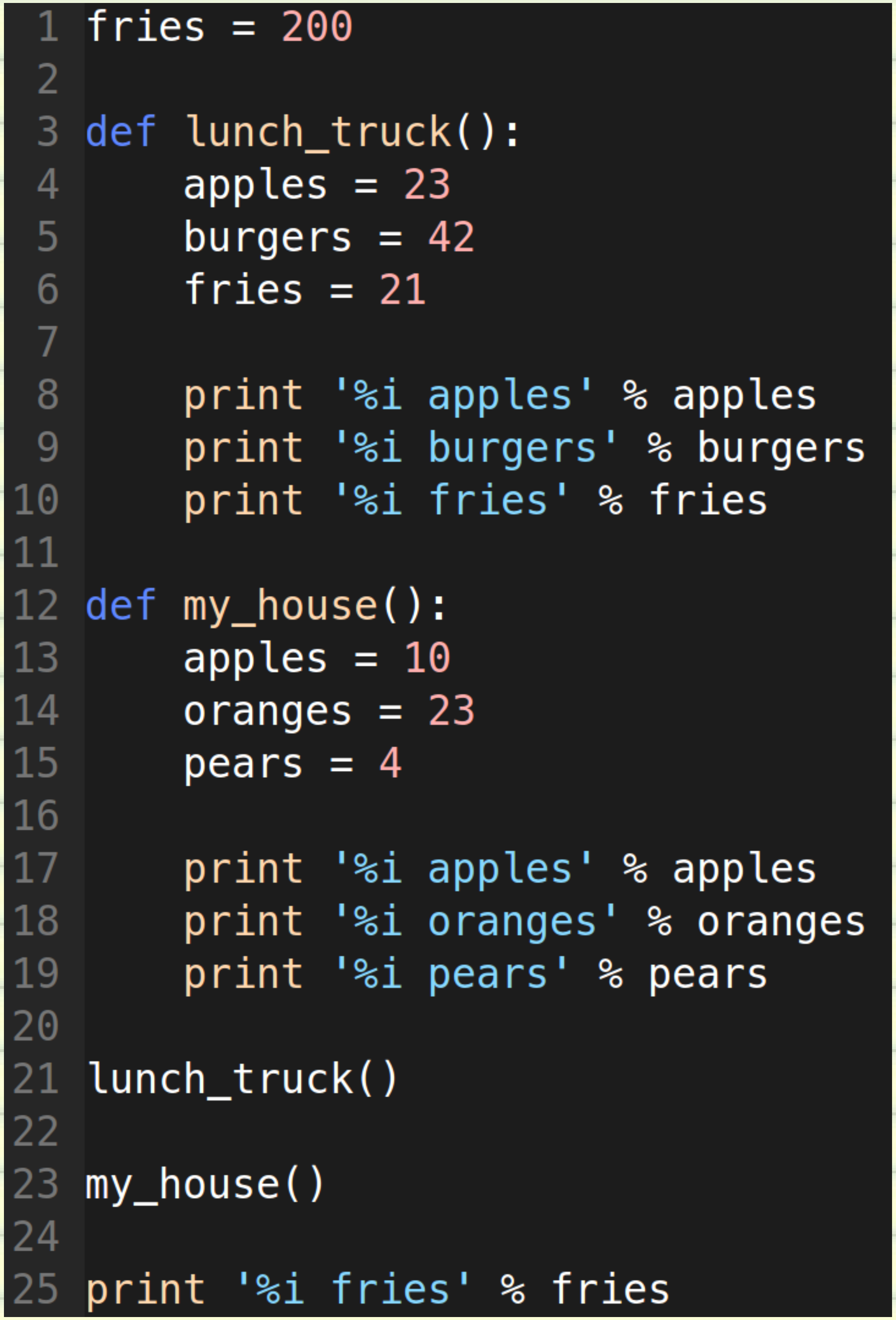


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Name search looks like this:

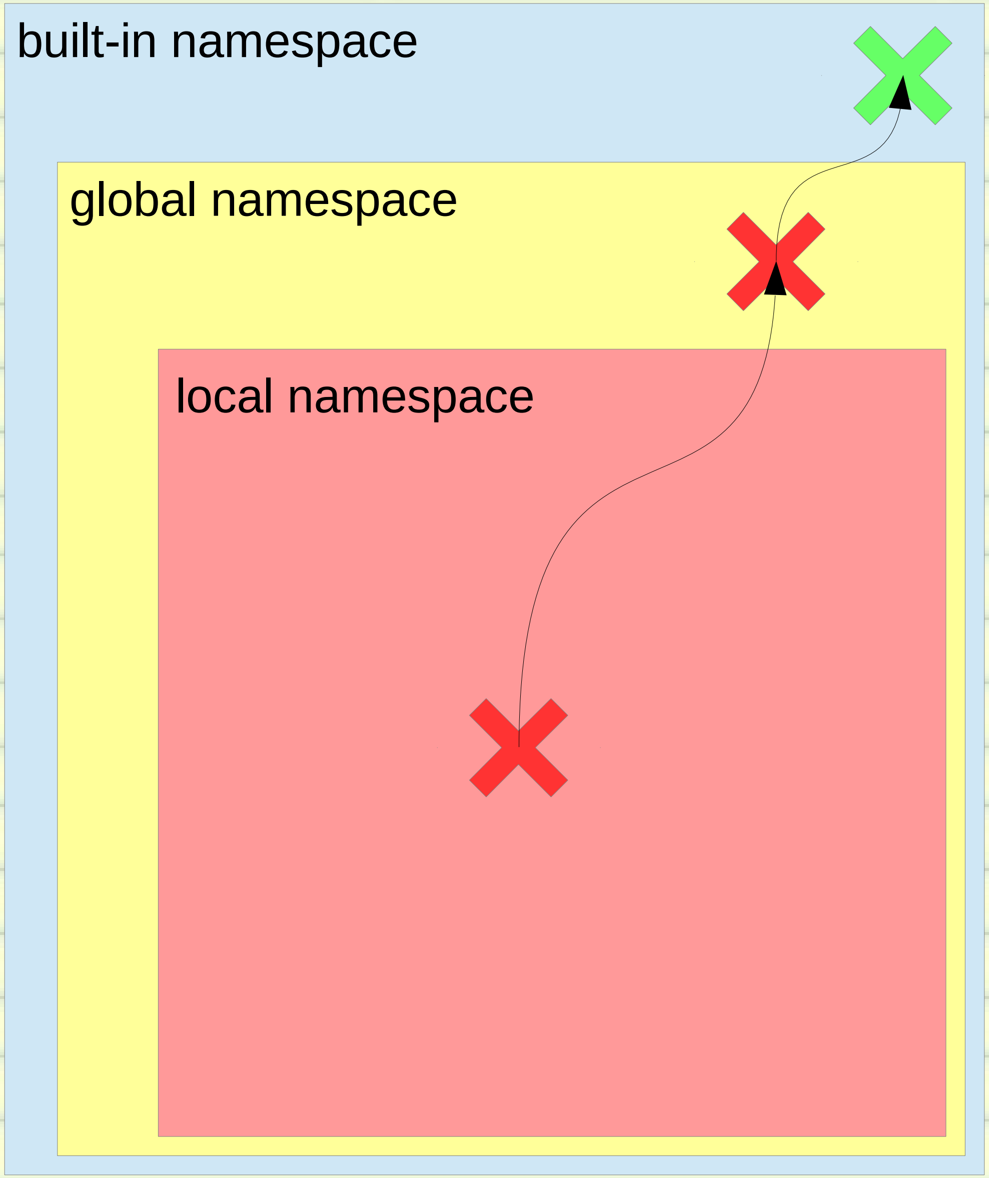


Namespaces & Variable Scope

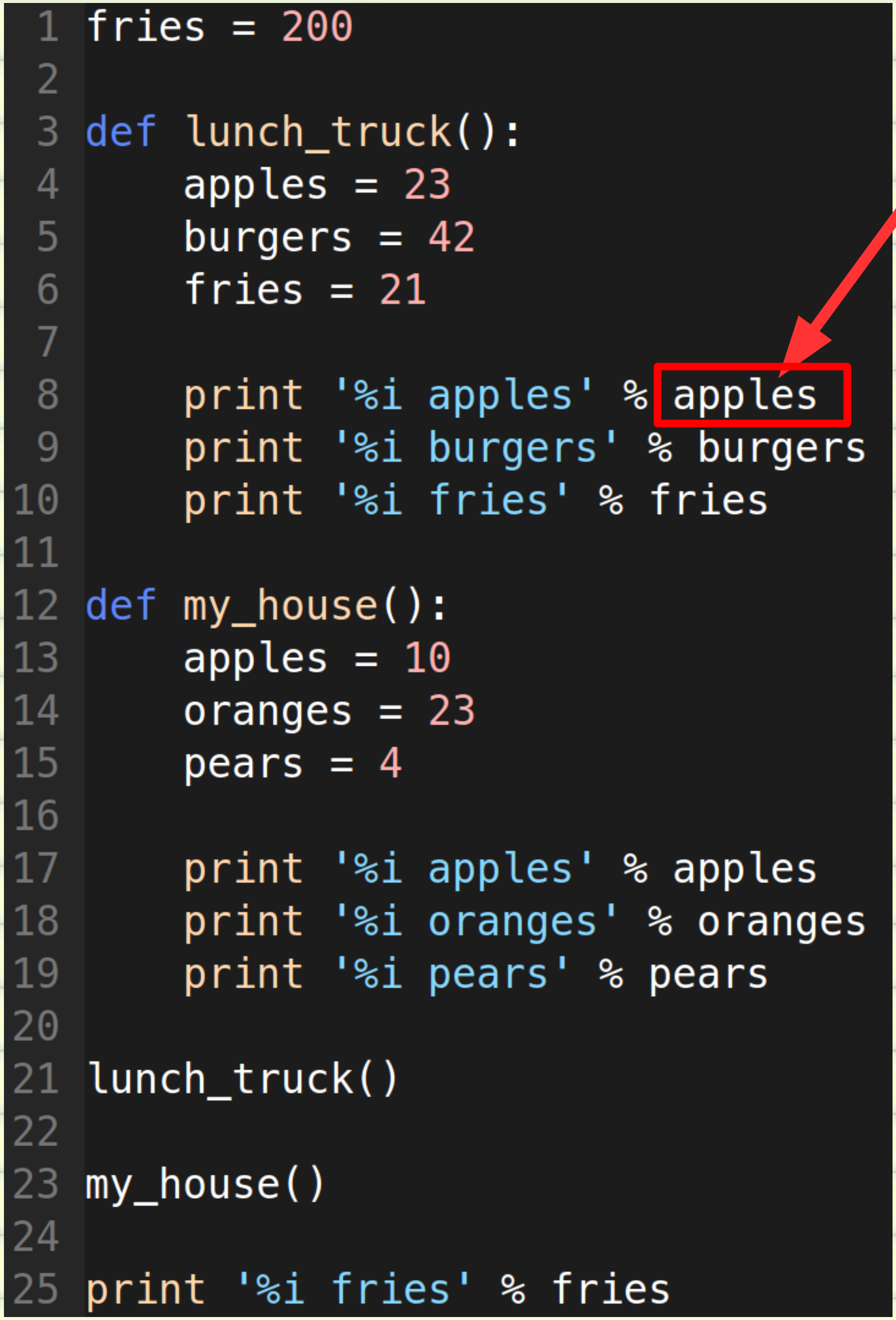


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Name search looks like this:

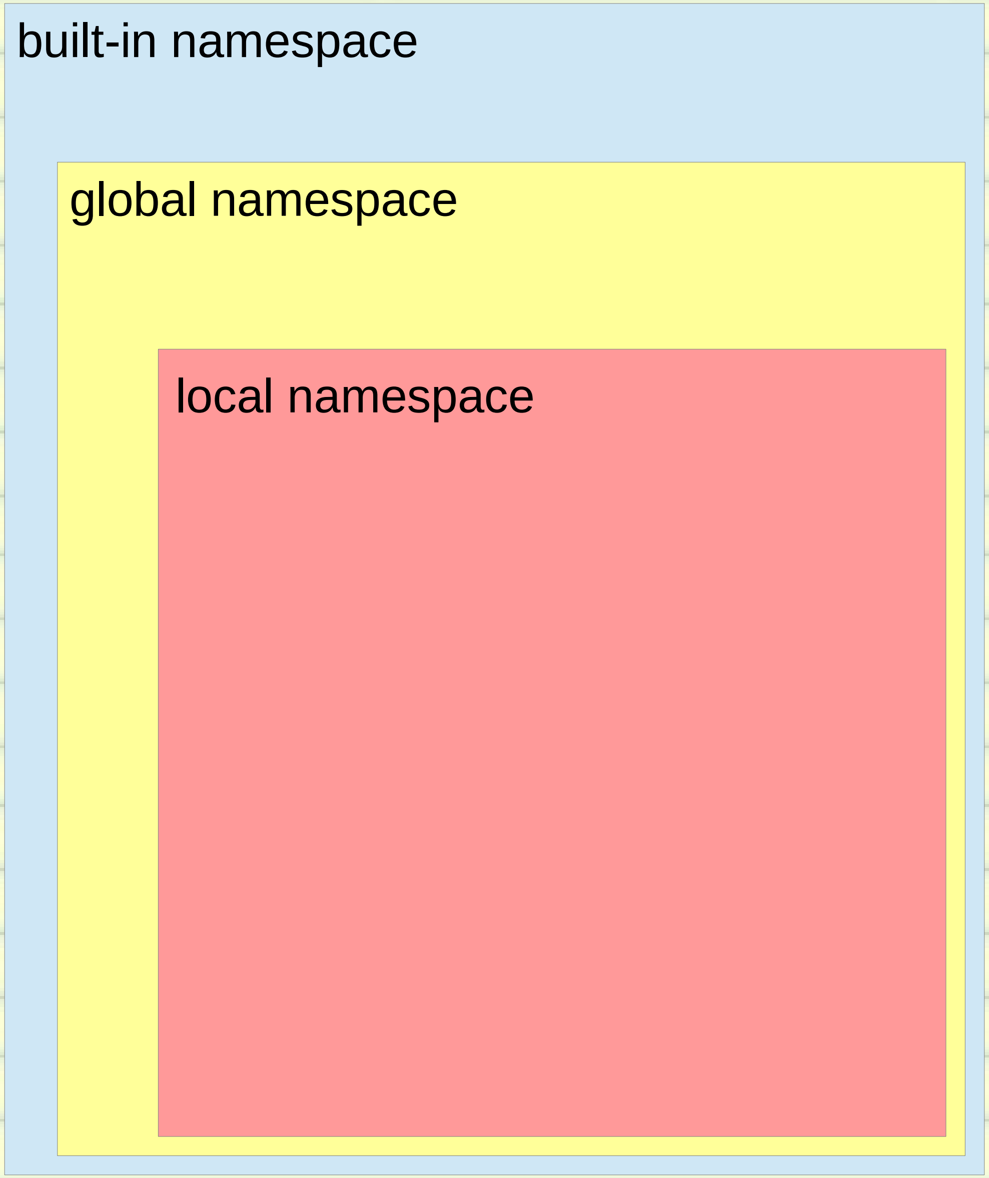


Namespaces & Variable Scope

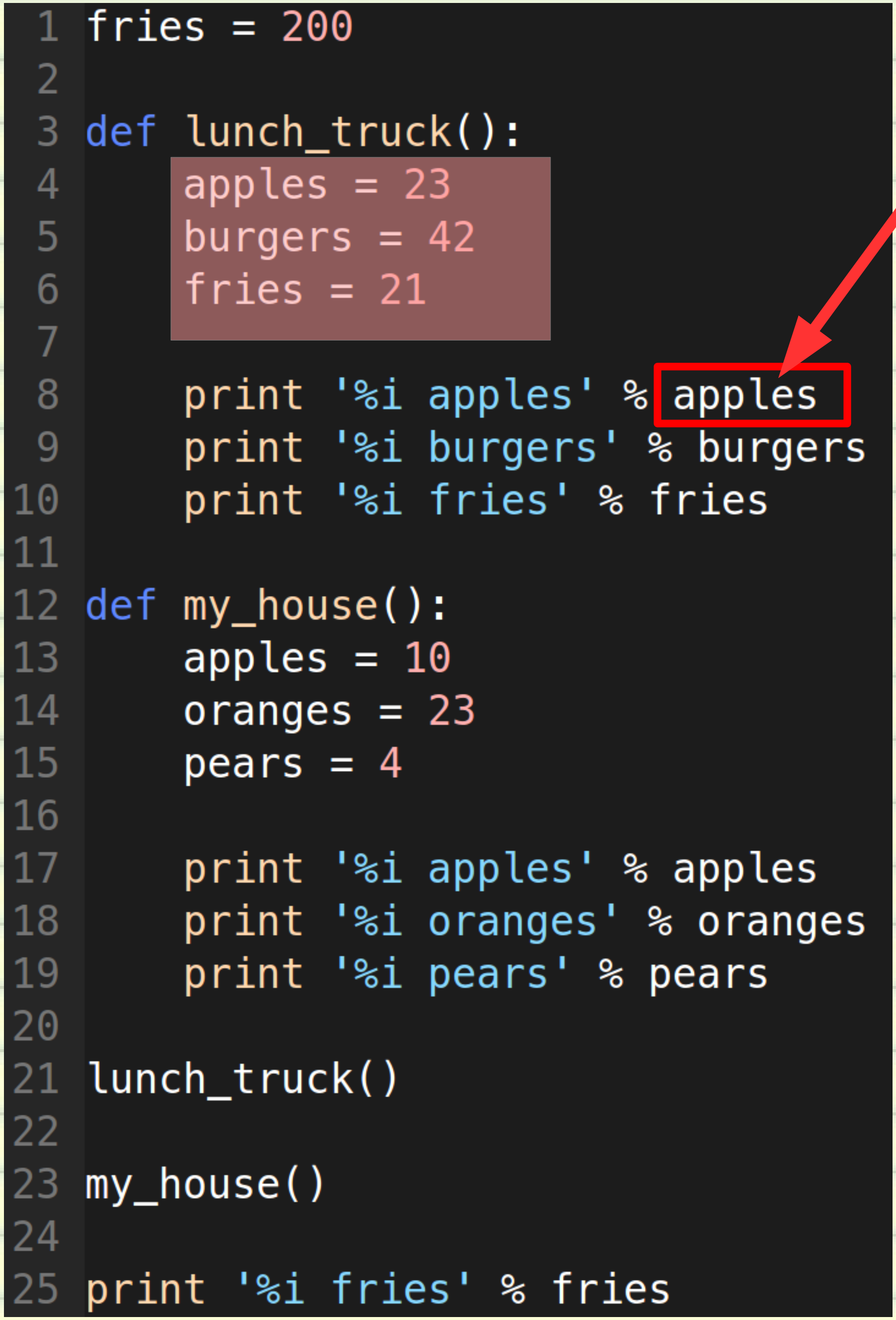


```python
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Let's search for this.

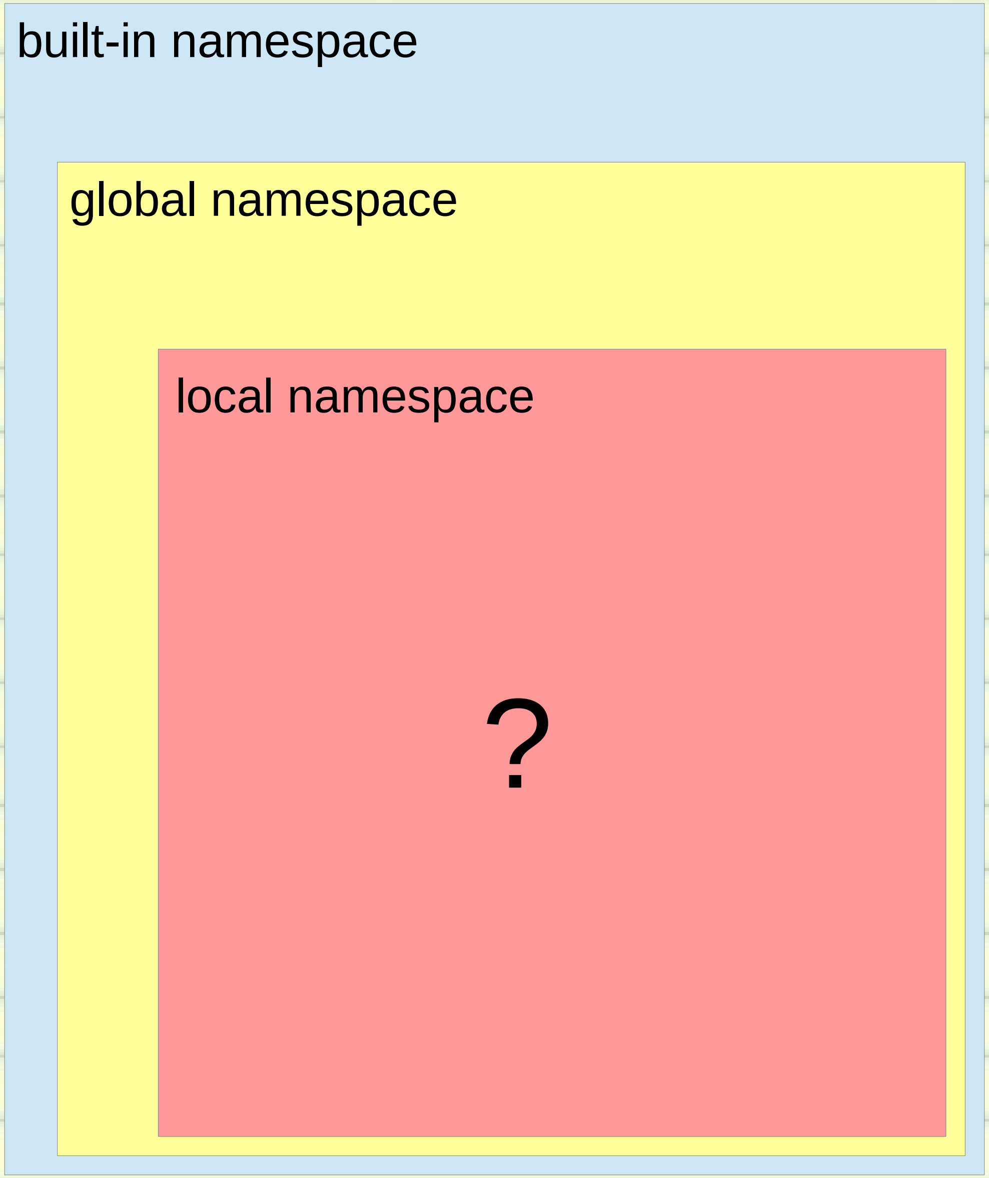


Namespaces & Variable Scope

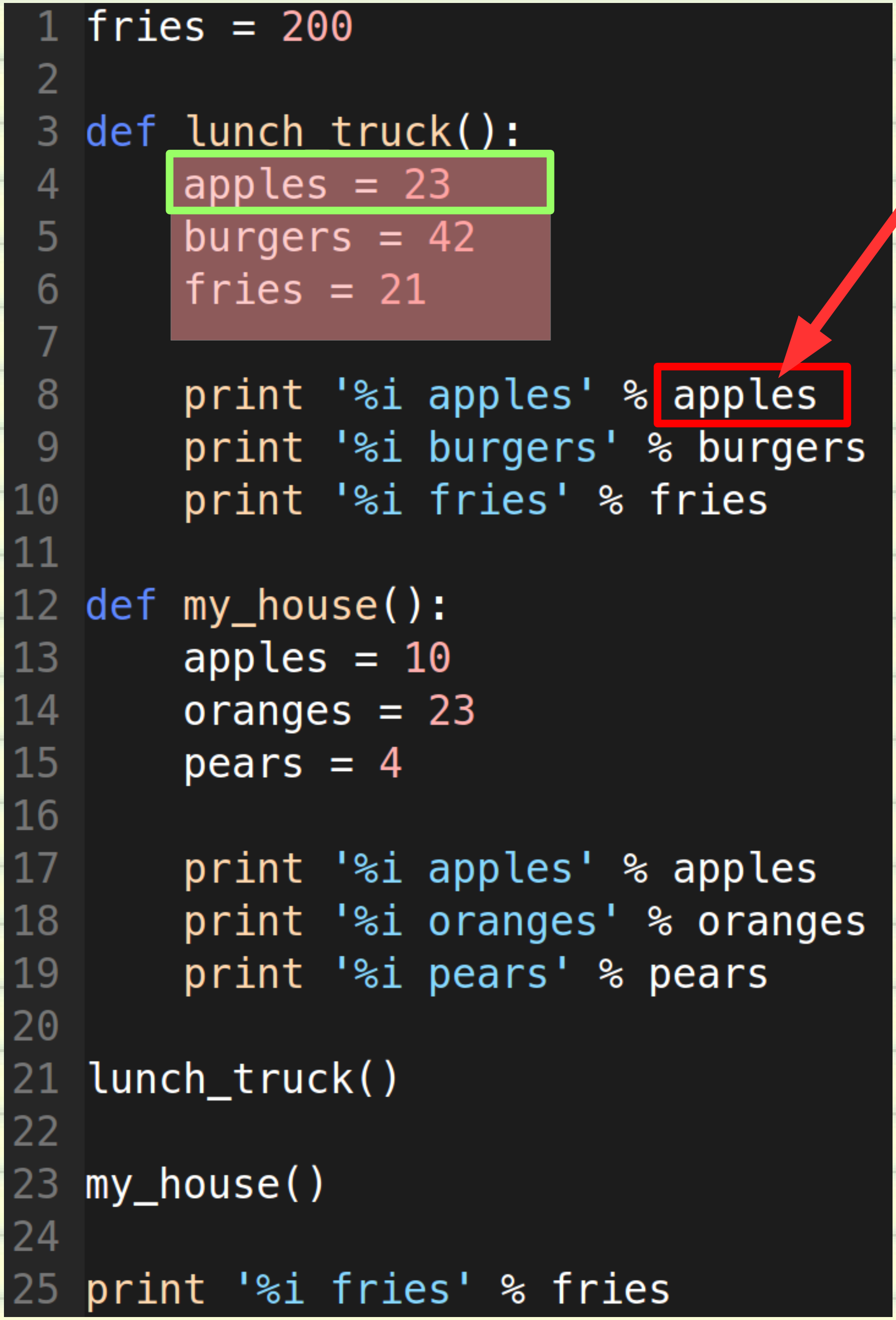


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Let's search for this.

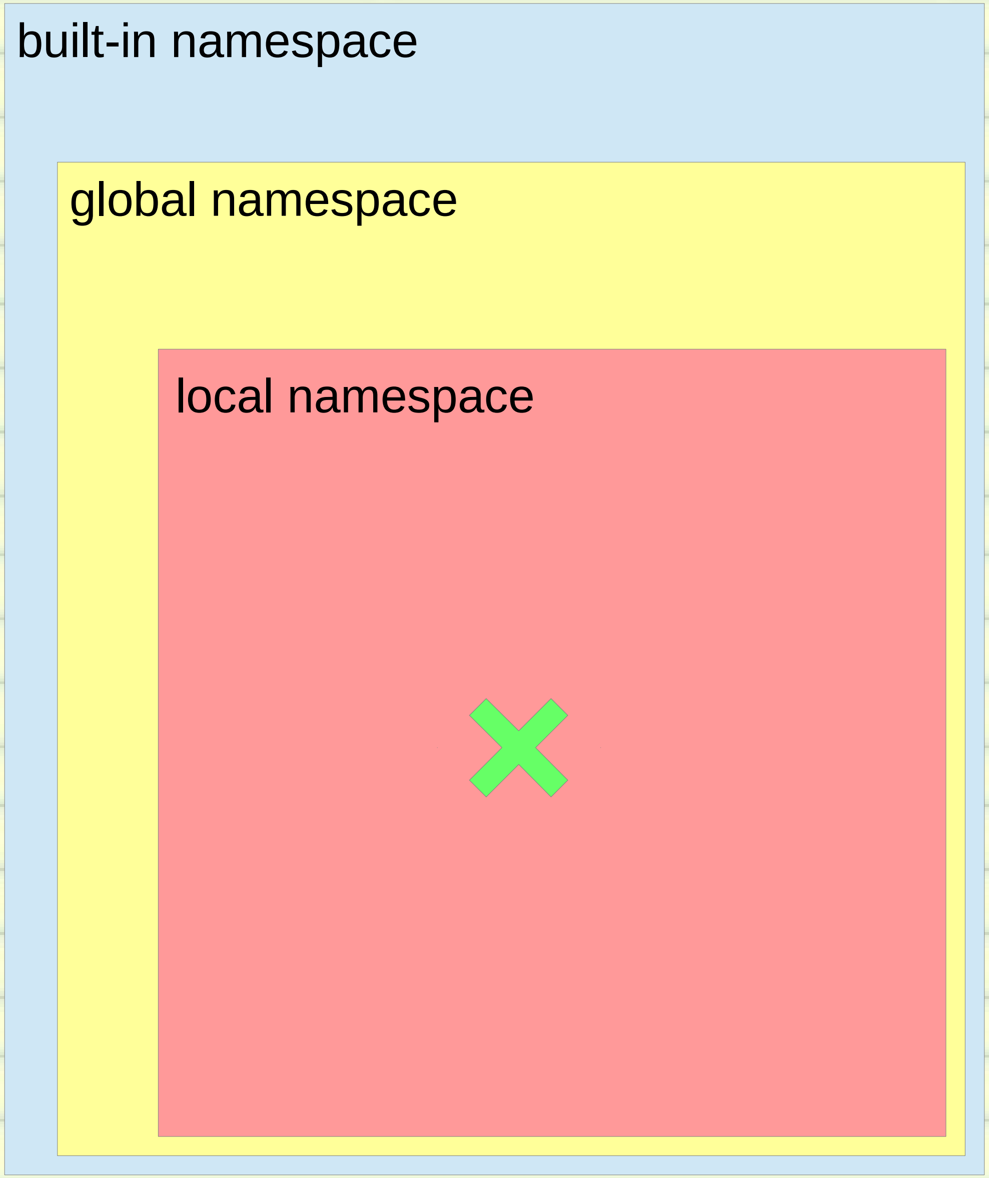


Namespaces & Variable Scope

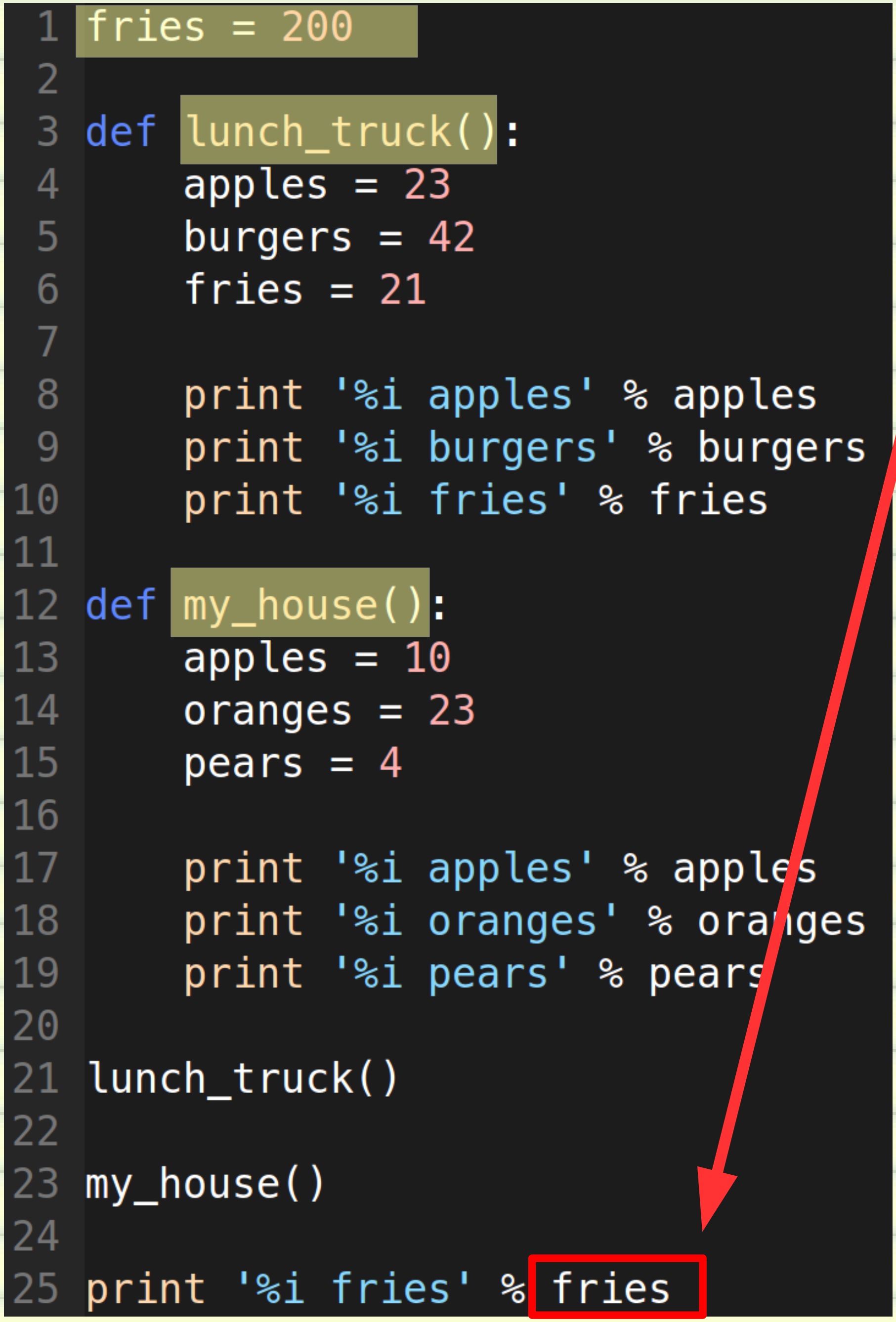


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Let's search for this.

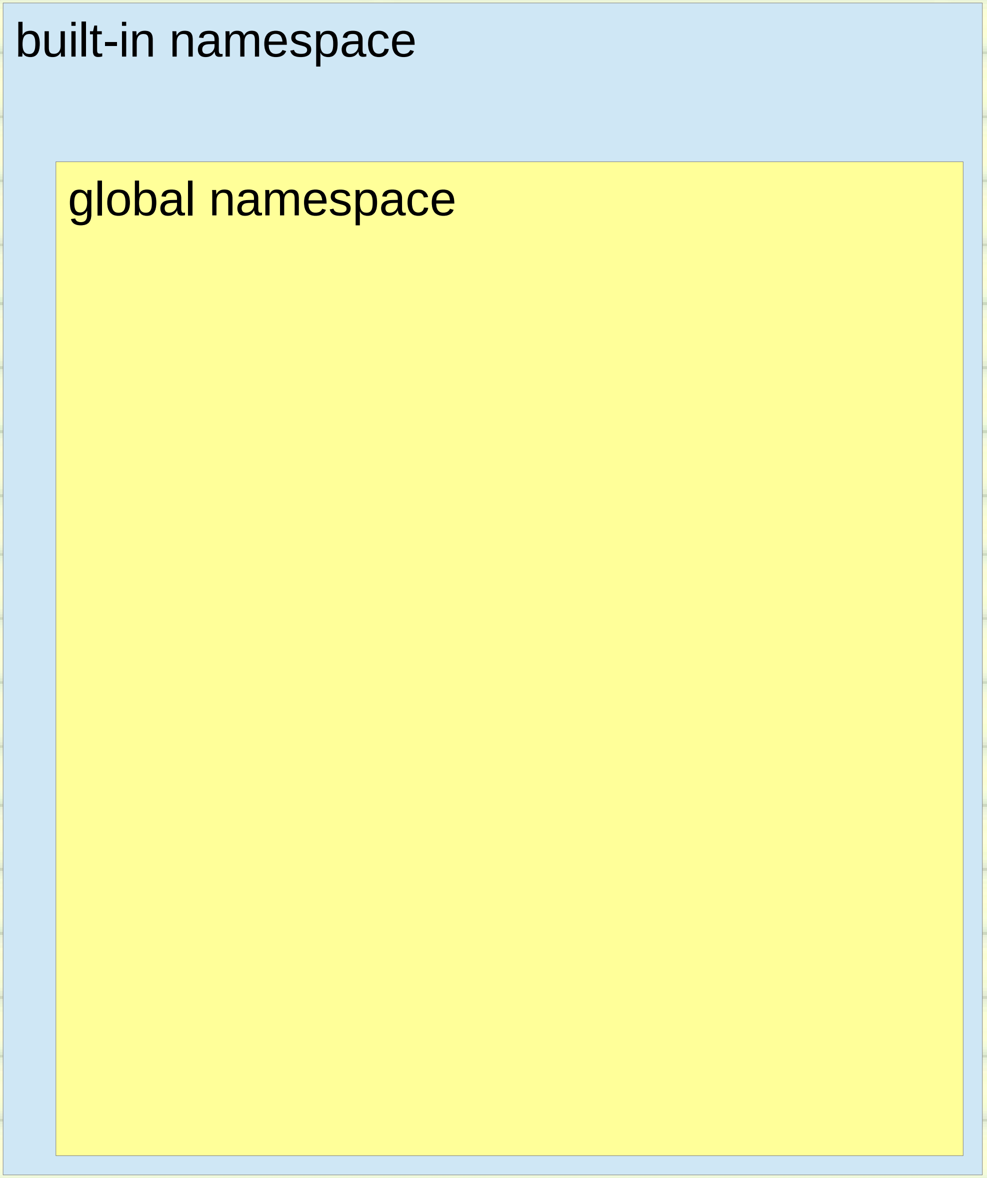


Namespaces & Variable Scope

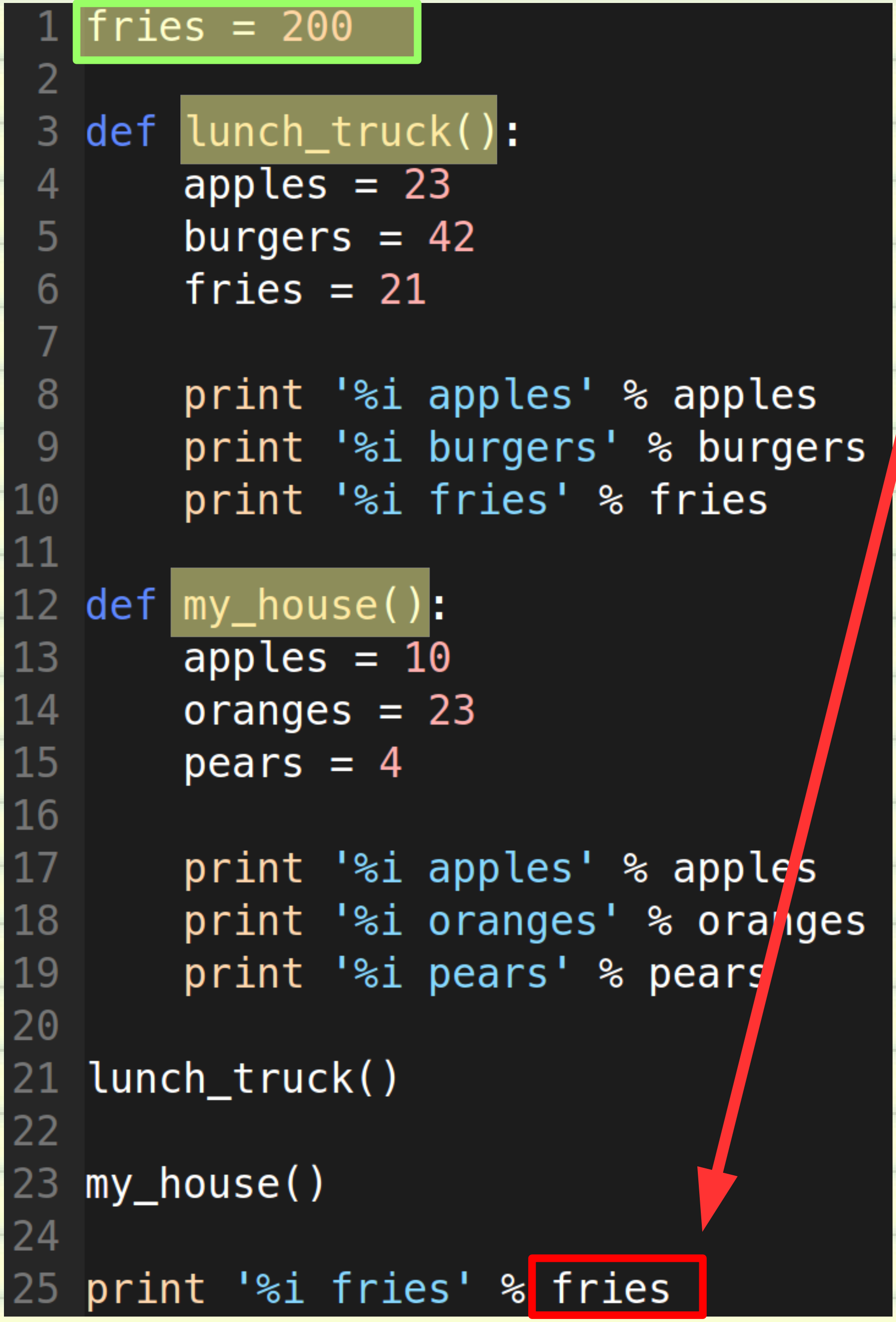


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Let's search for this.

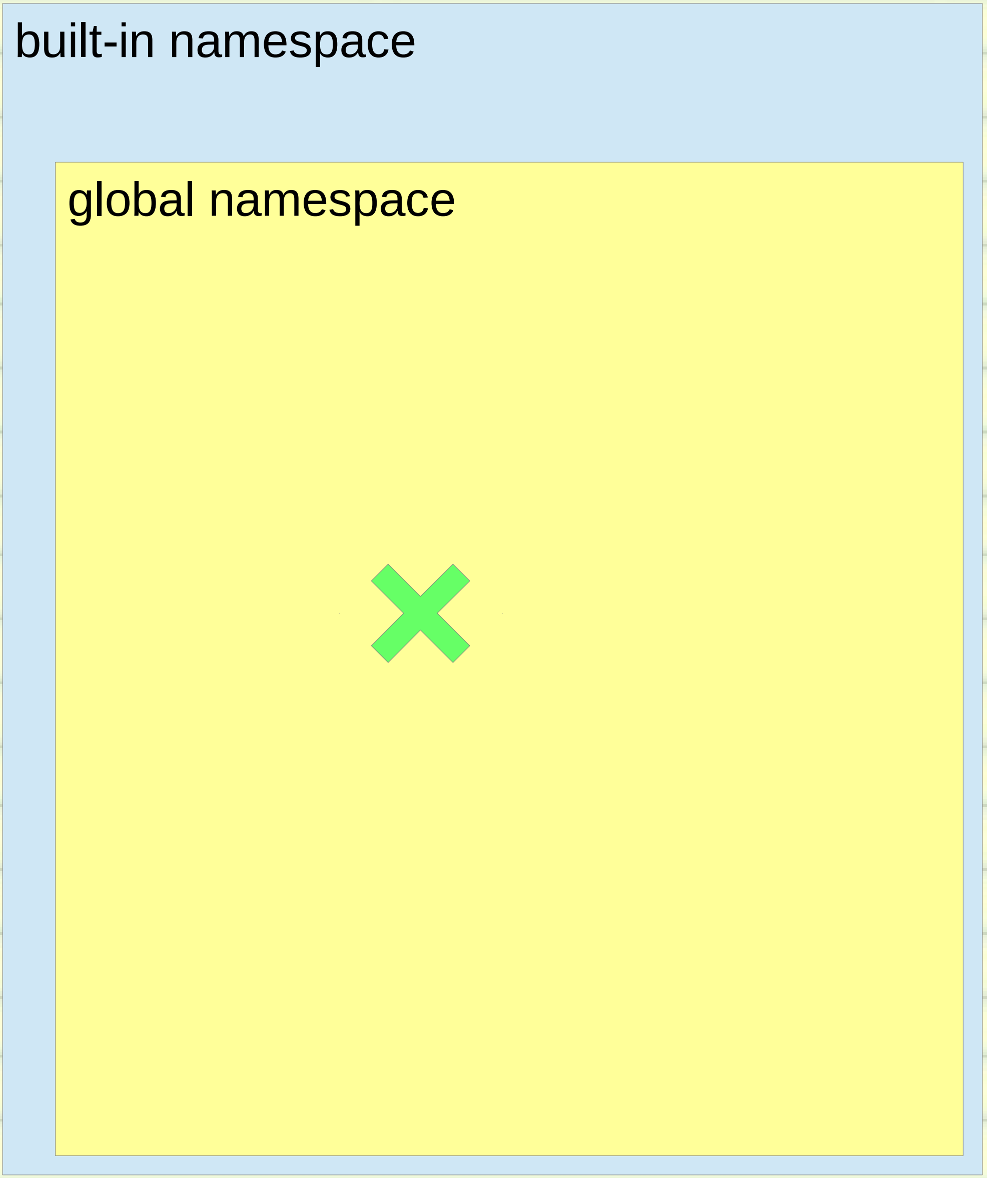


Namespaces & Variable Scope

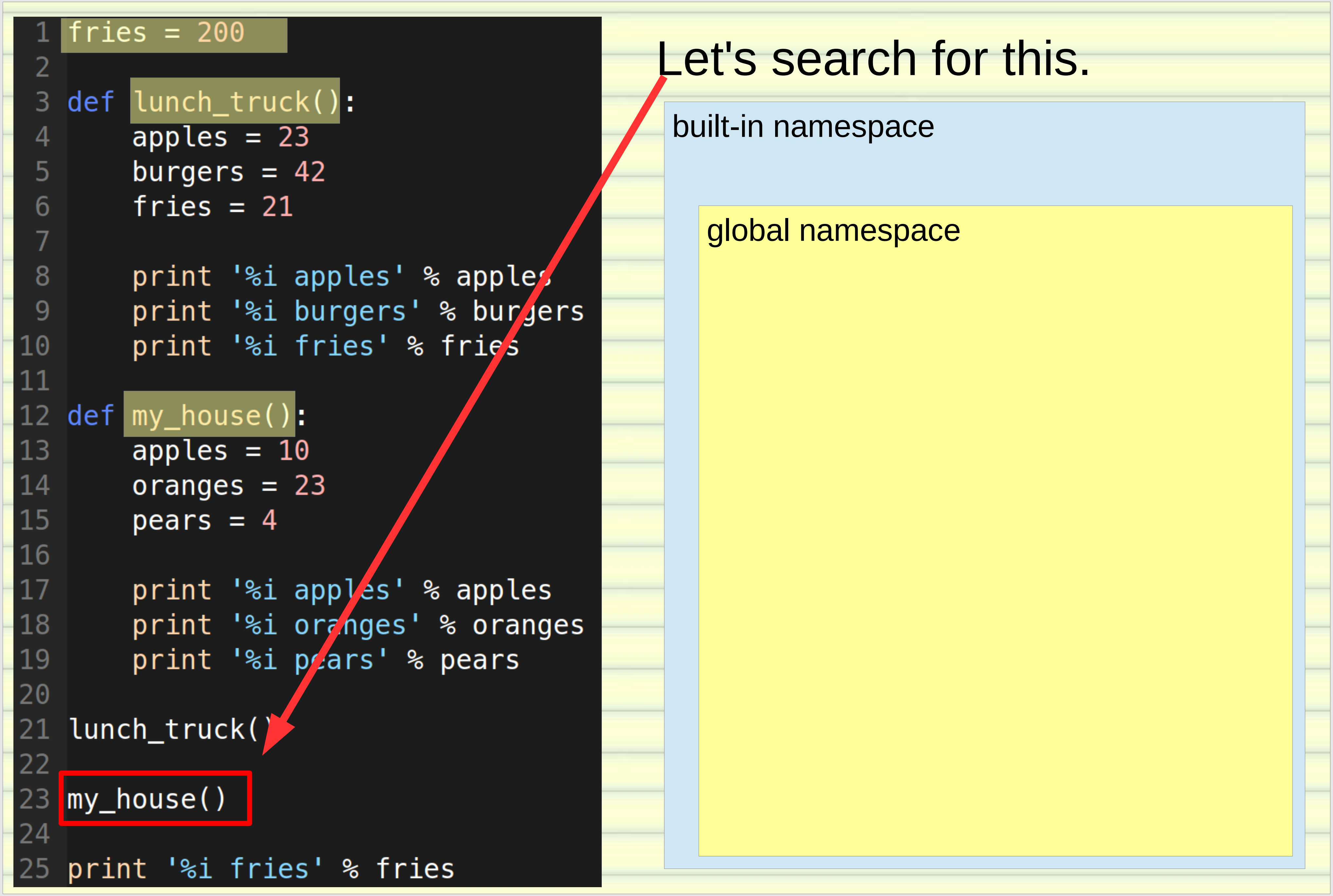


```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

Let's search for this.



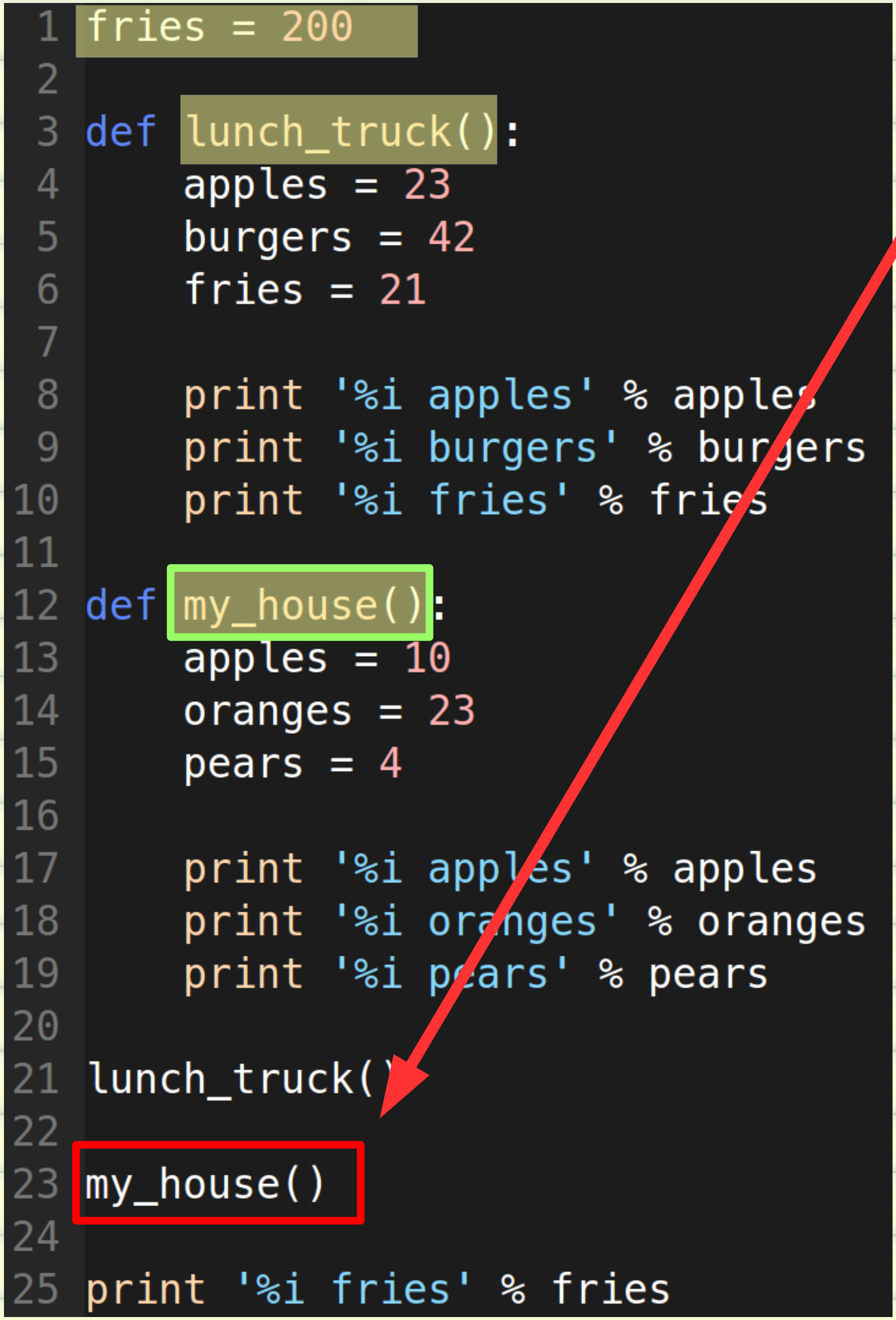
Namespaces & Variable Scope



```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

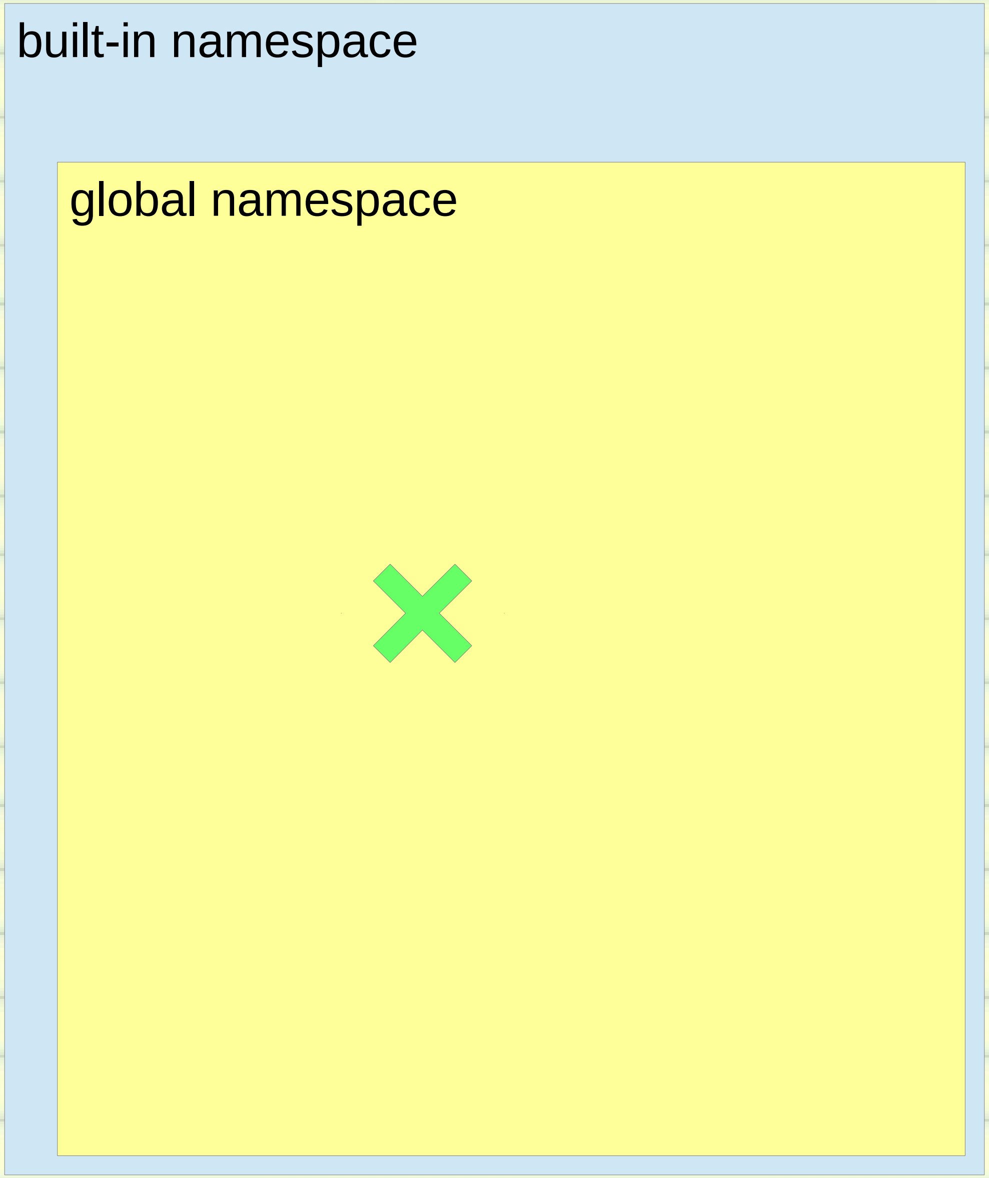
Let's search for this.

Namespaces & Variable Scope



```
fries = 200

def lunch_truck():
    apples = 23
    burgers = 42
    fries = 21

    print '%i apples' % apples
    print '%i burgers' % burgers
    print '%i fries' % fries

def my_house():
    apples = 10
    oranges = 23
    pears = 4

    print '%i apples' % apples
    print '%i oranges' % oranges
    print '%i pears' % pears

lunch_truck()

my_house()

print '%i fries' % fries
```

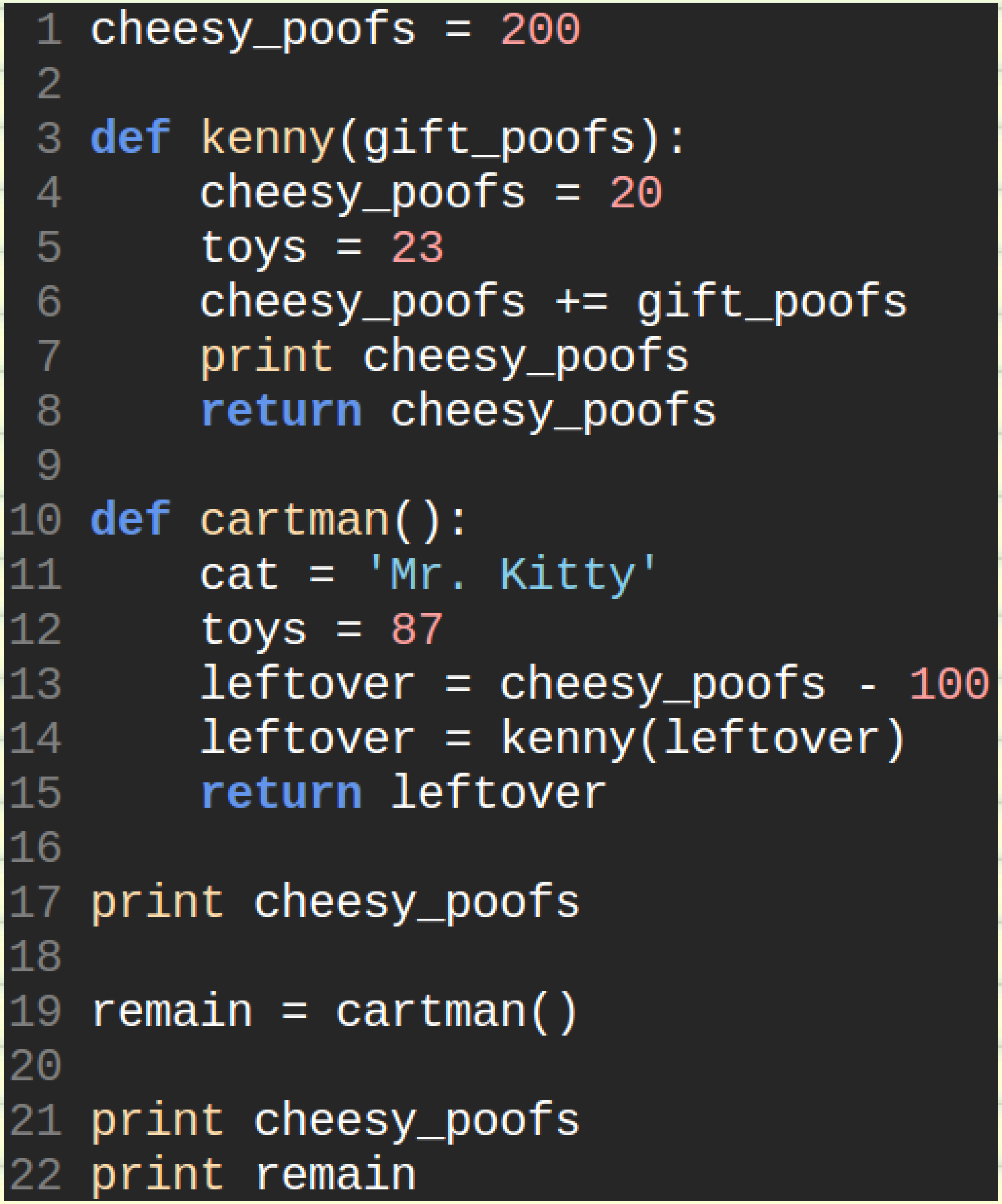
Let's search for this.



built-in namespace

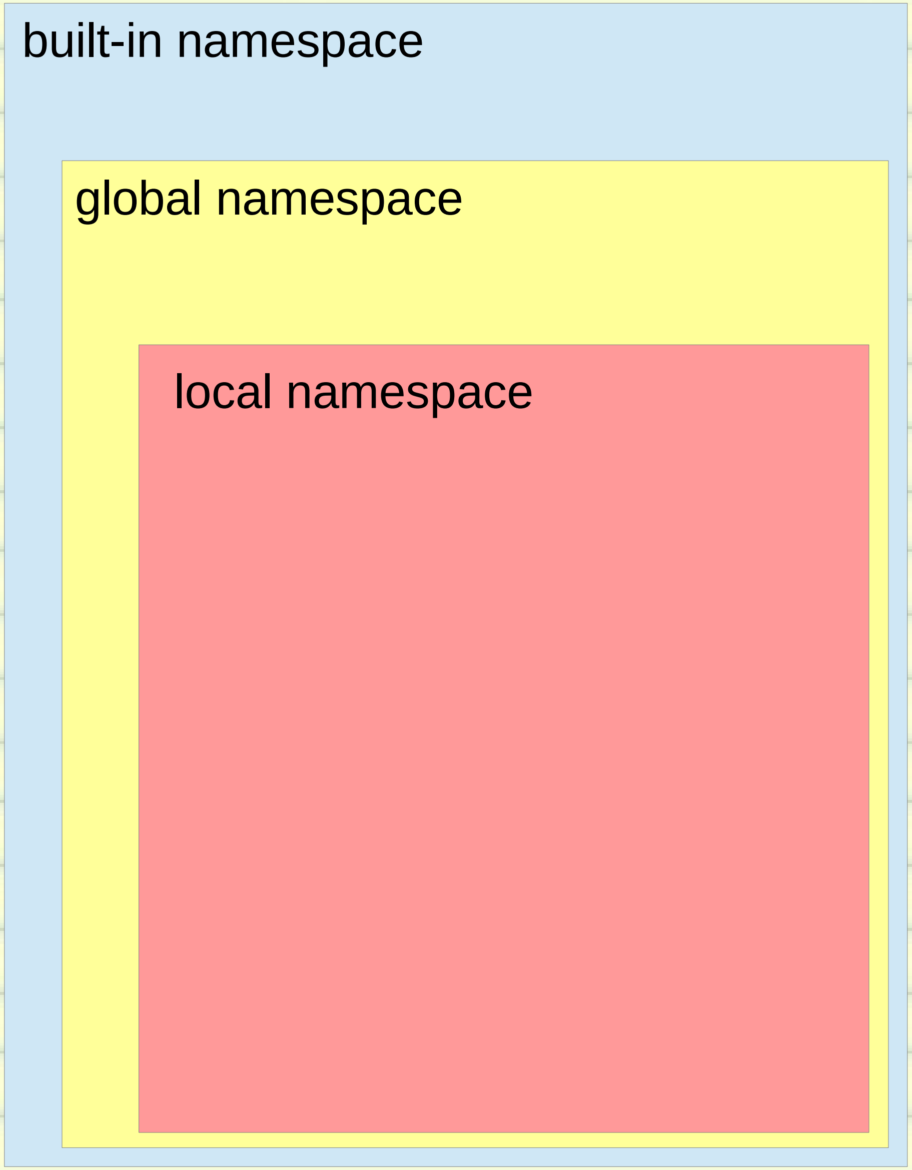
global namespace

Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

New Code.

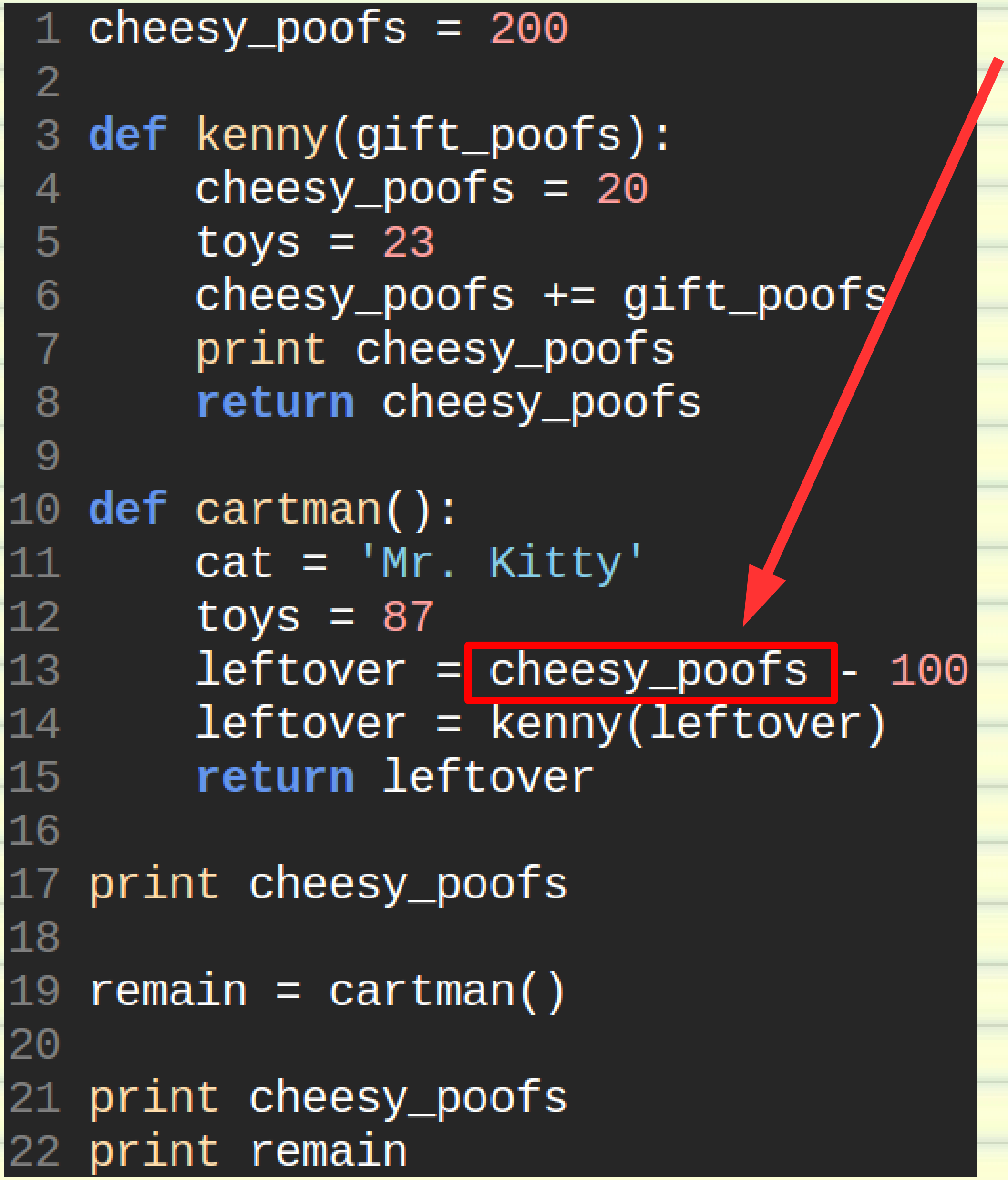


built-in namespace

global namespace

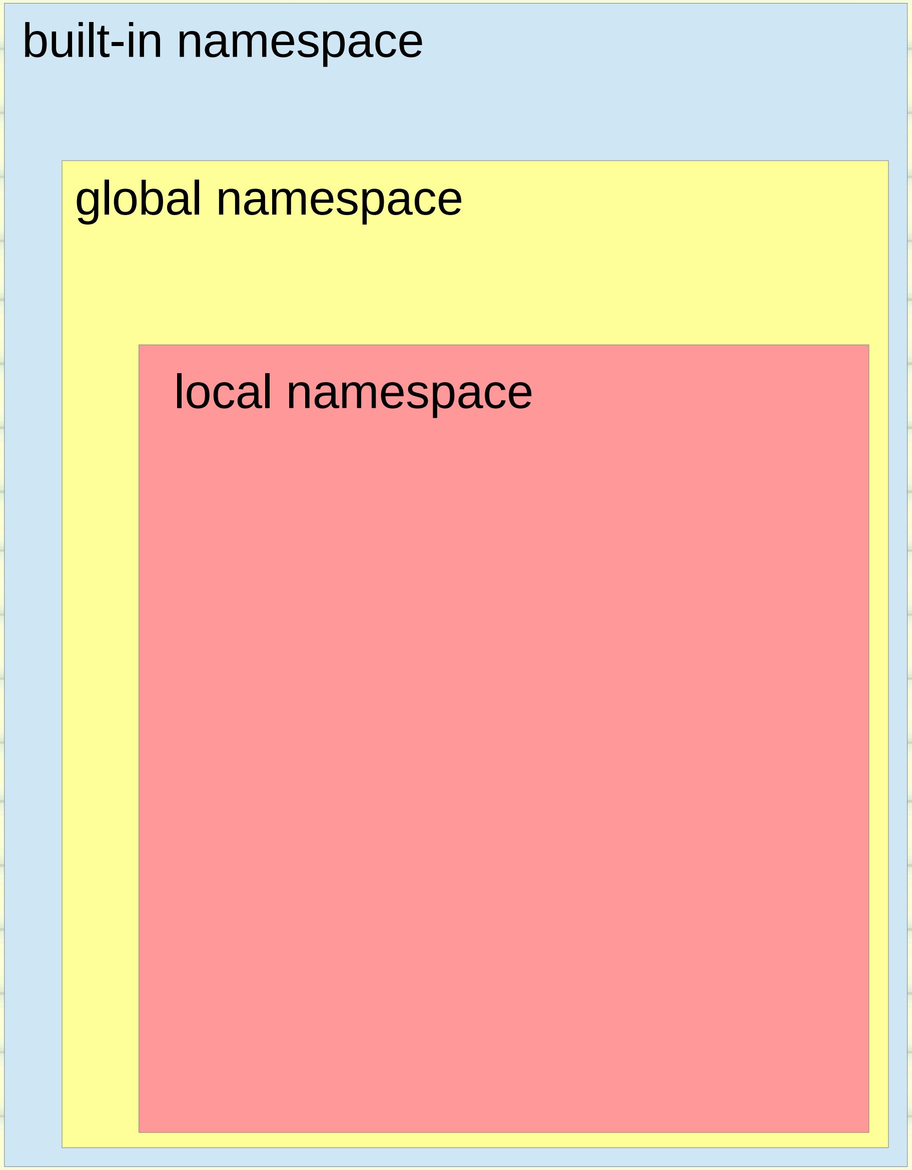
local namespace

Namespaces & Variable Scope

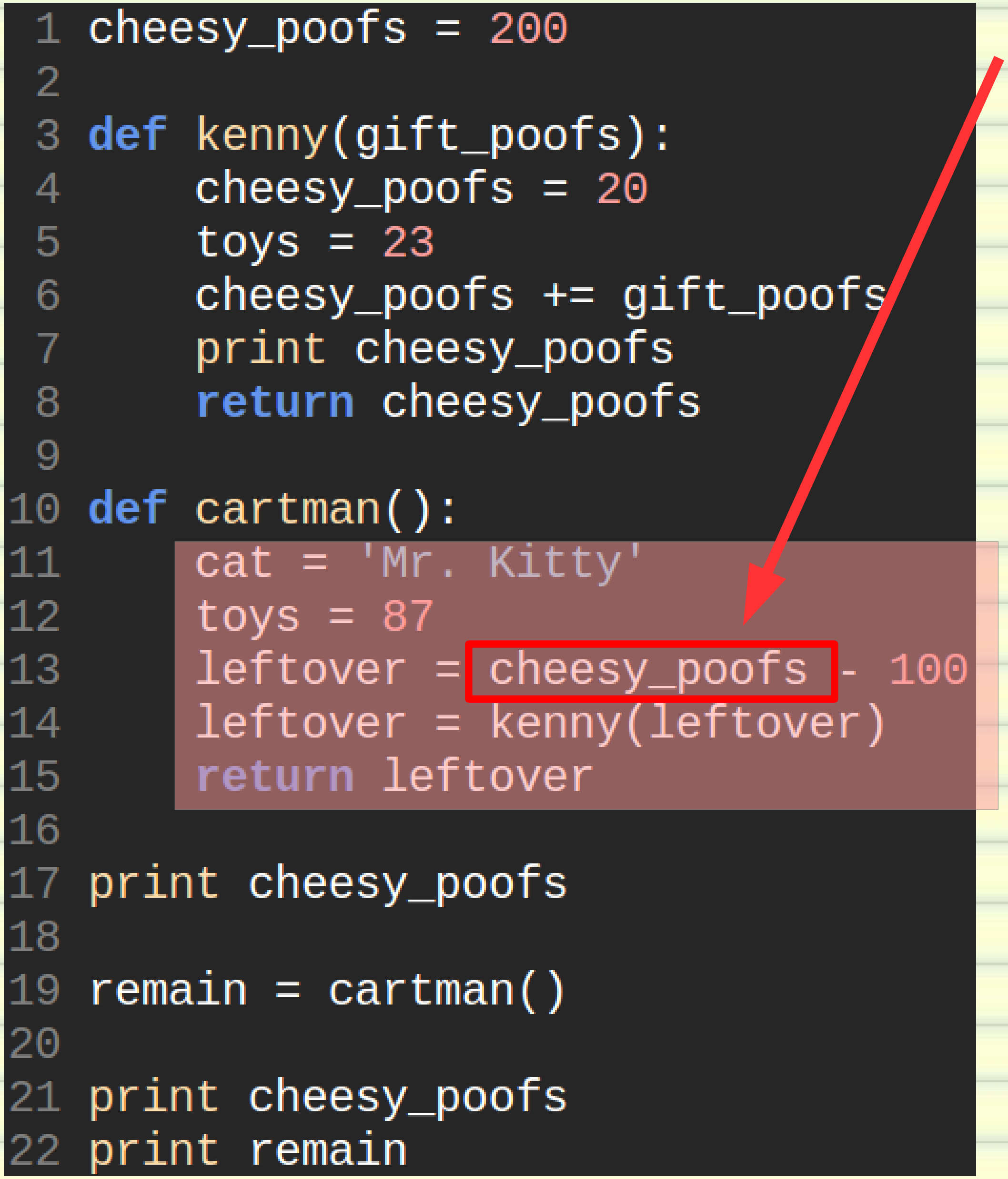


```python
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

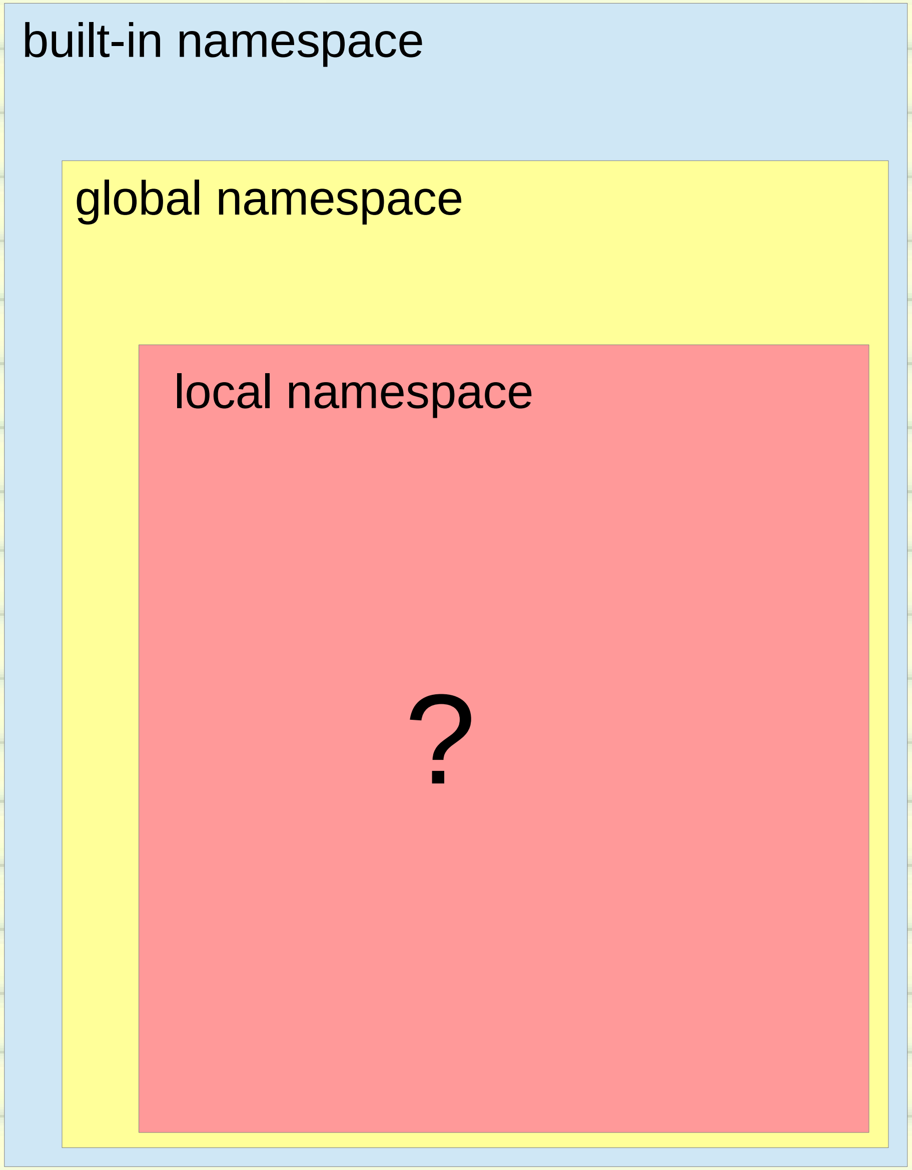


Namespaces & Variable Scope

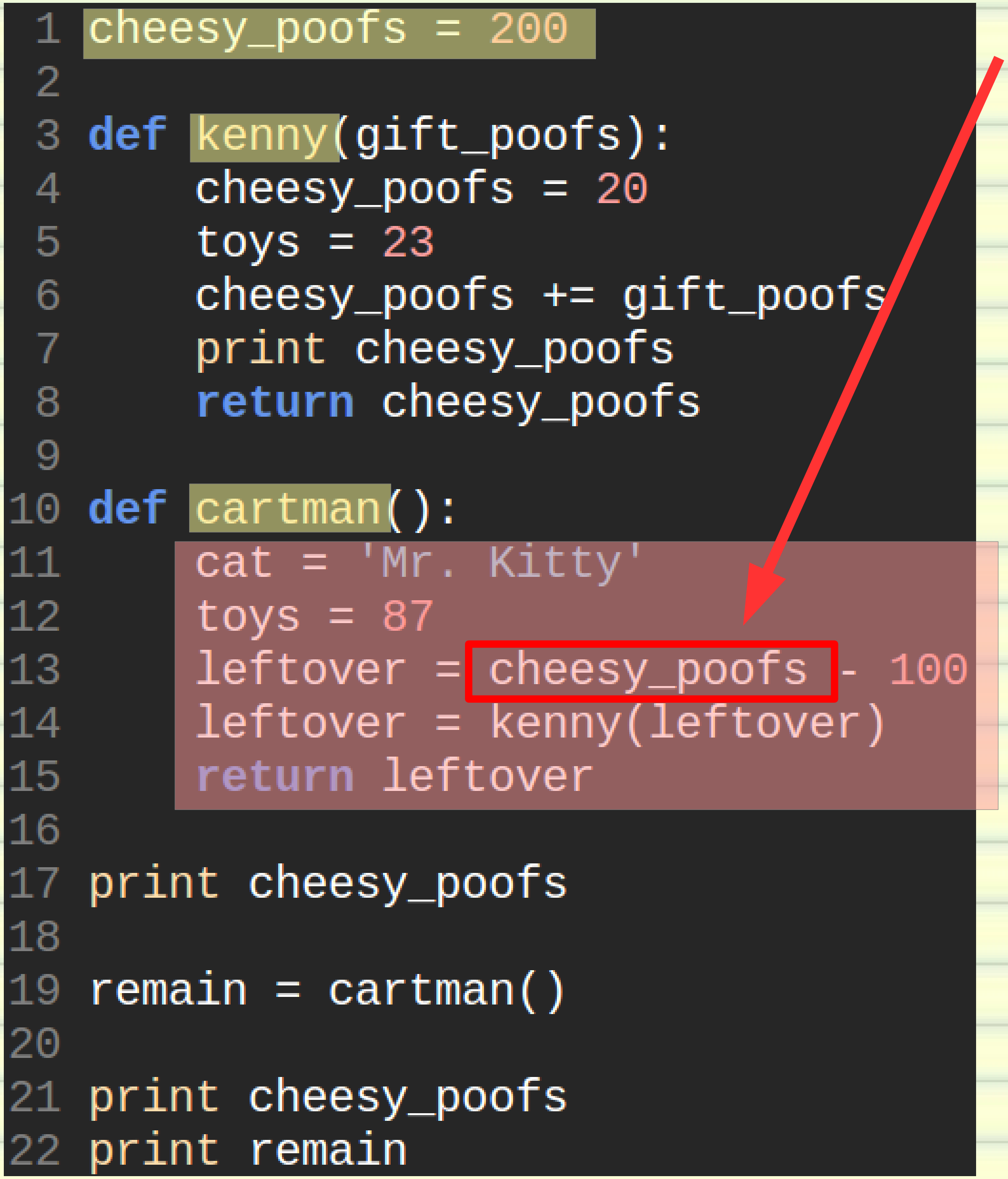


```python
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

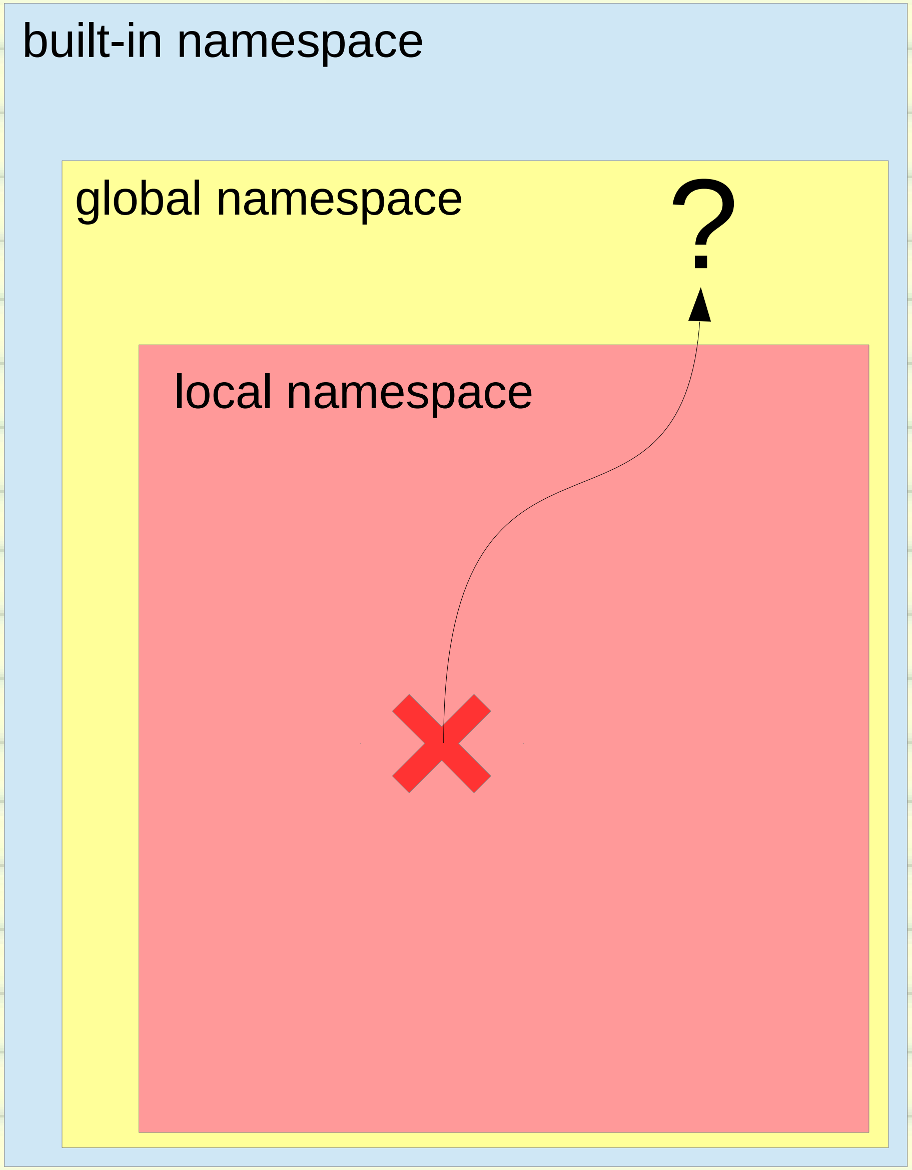


Namespaces & Variable Scope

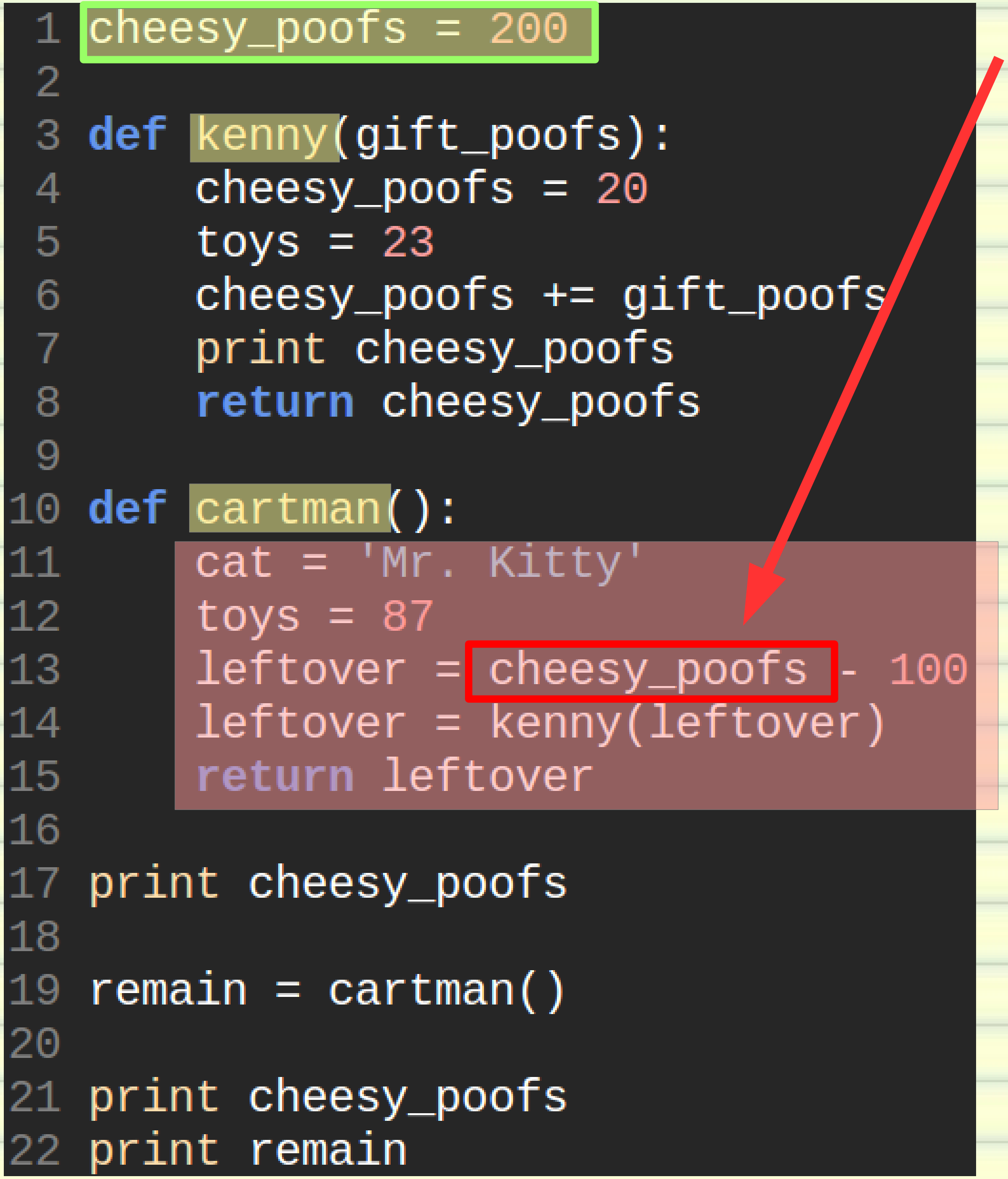


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

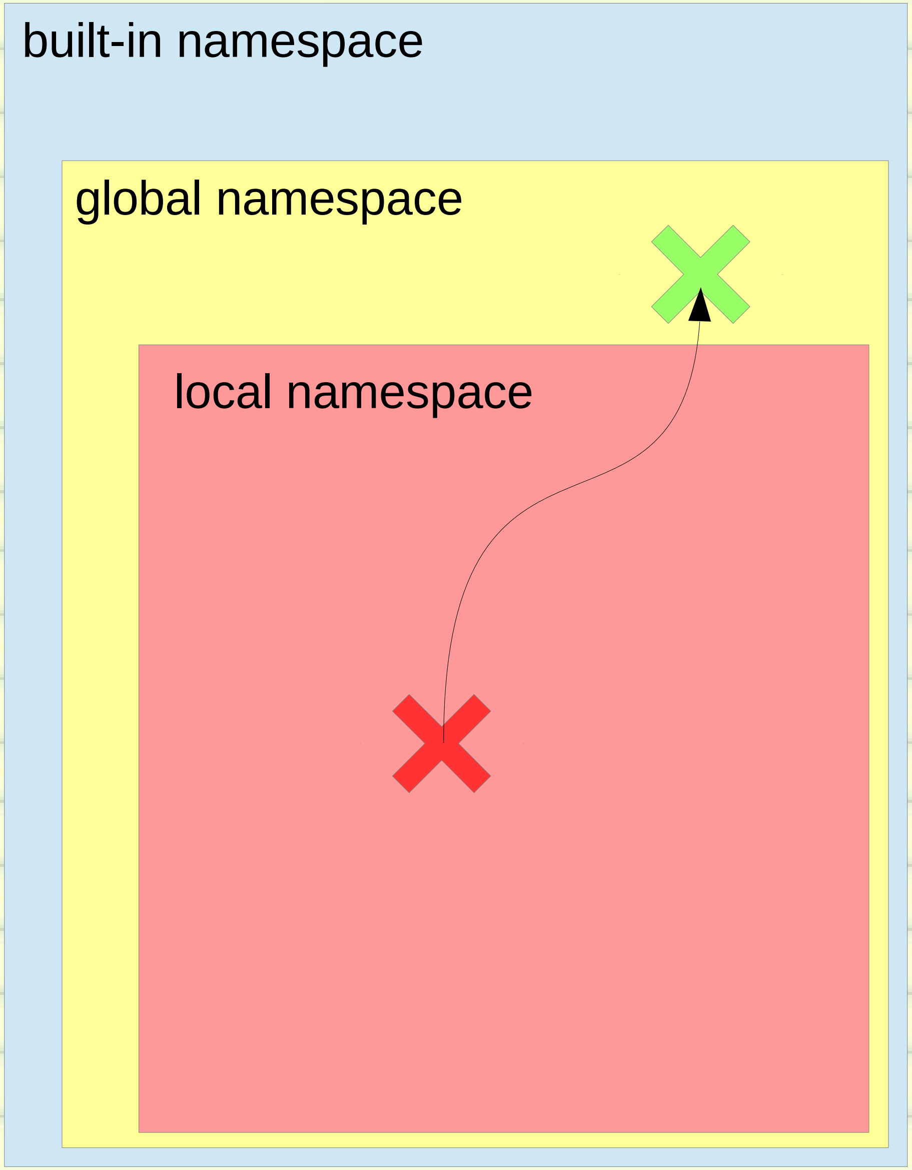


Namespaces & Variable Scope

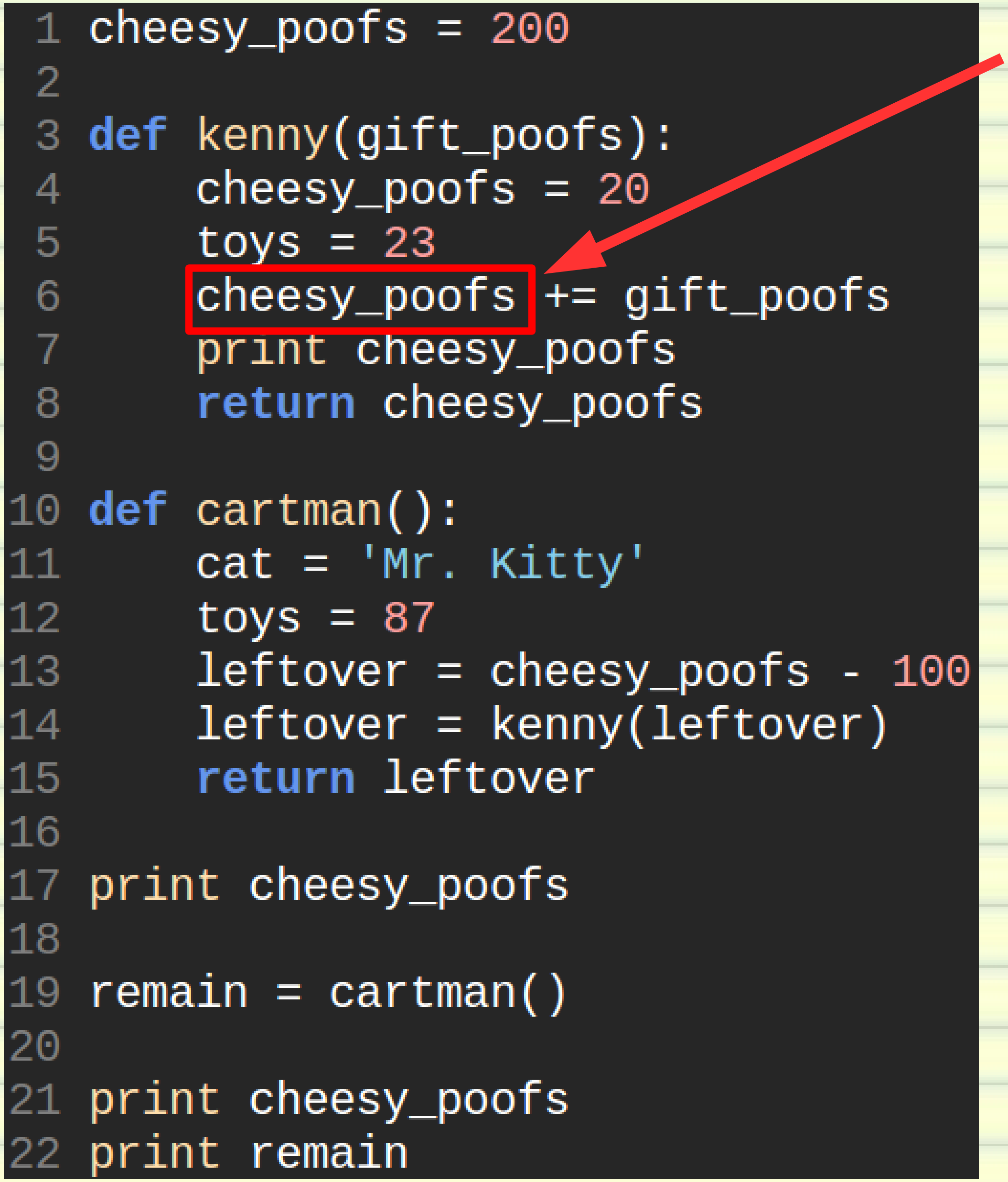


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

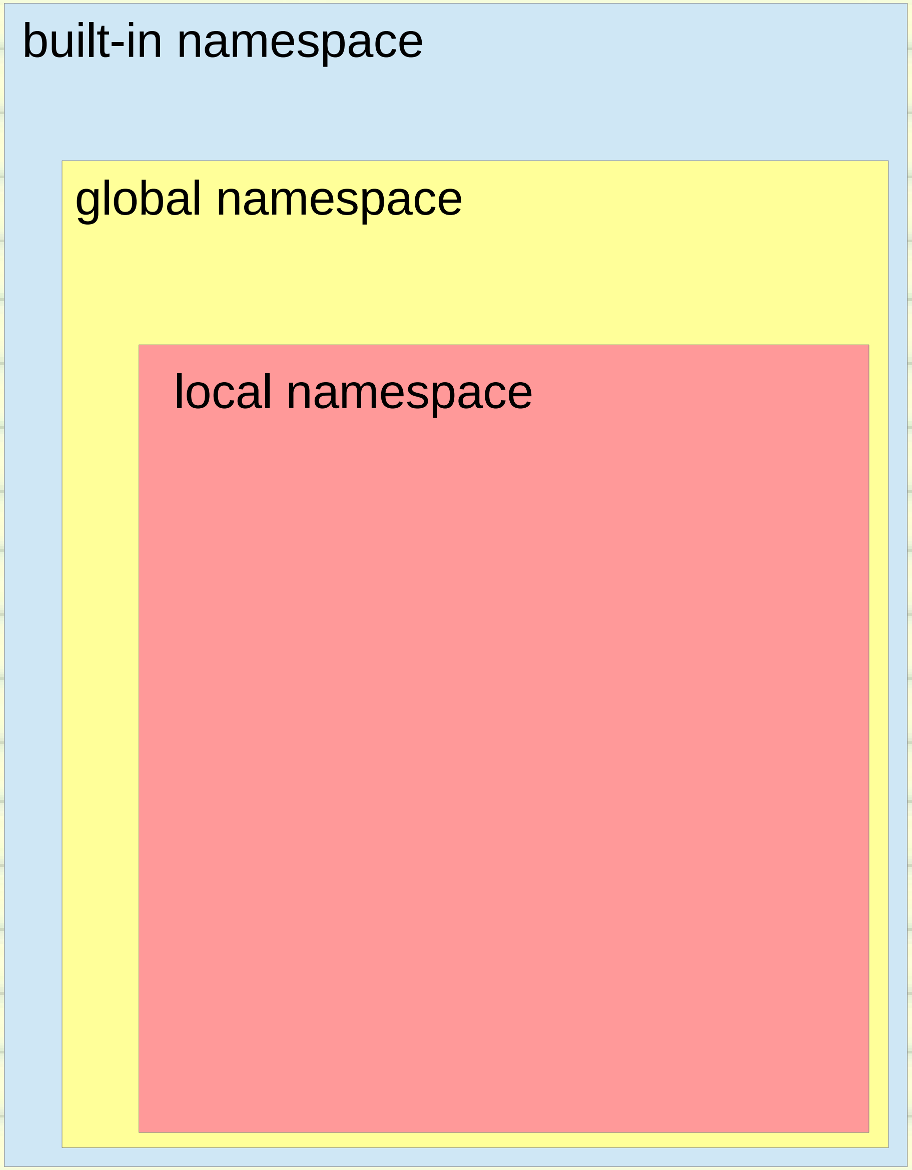


Namespaces & Variable Scope

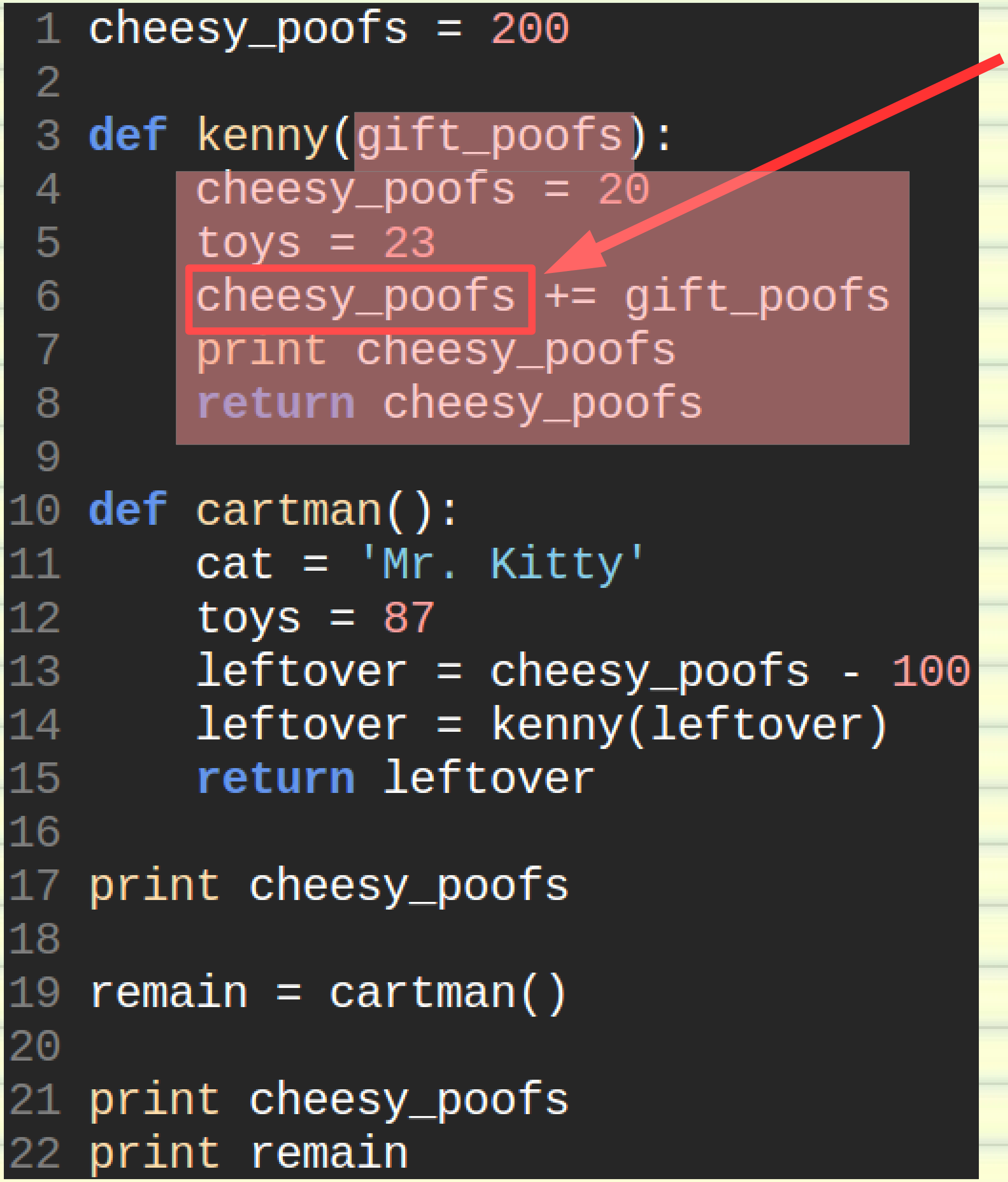


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

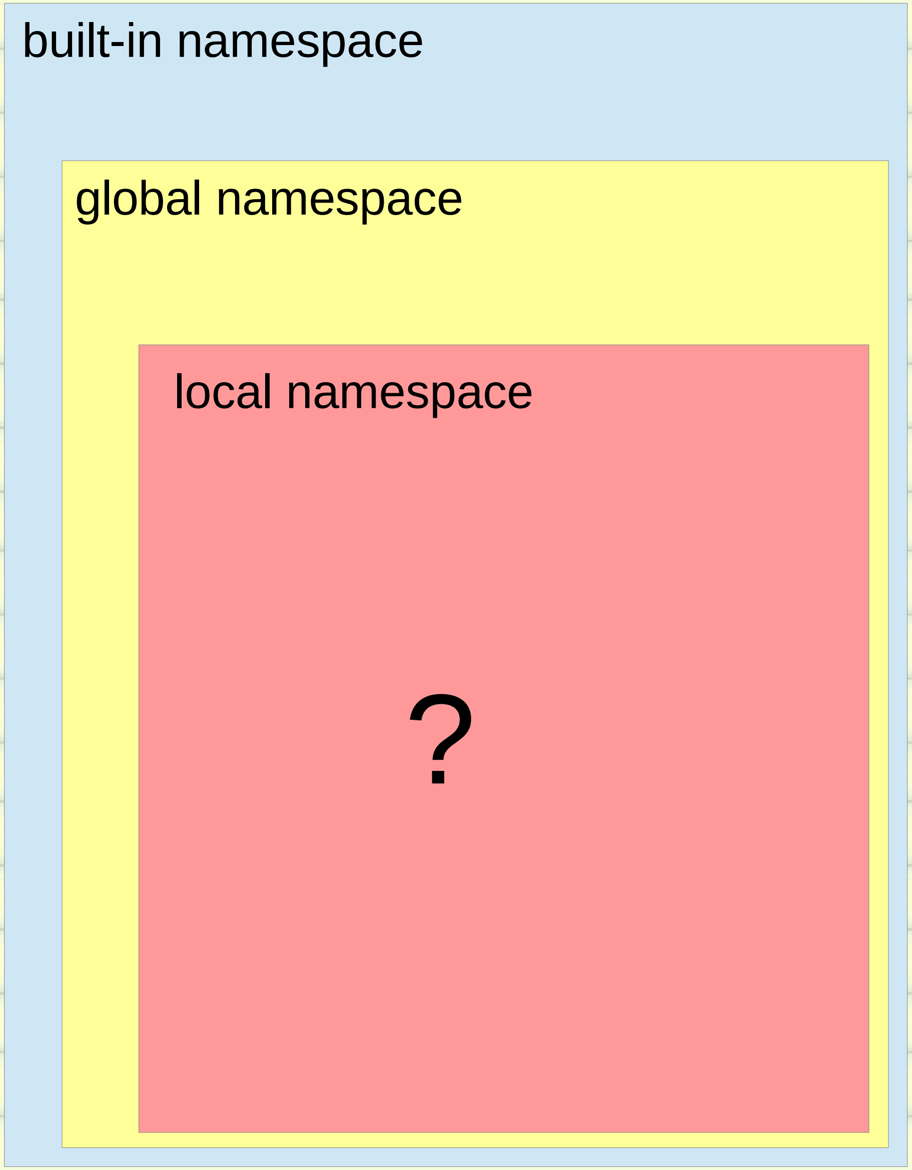


Namespaces & Variable Scope

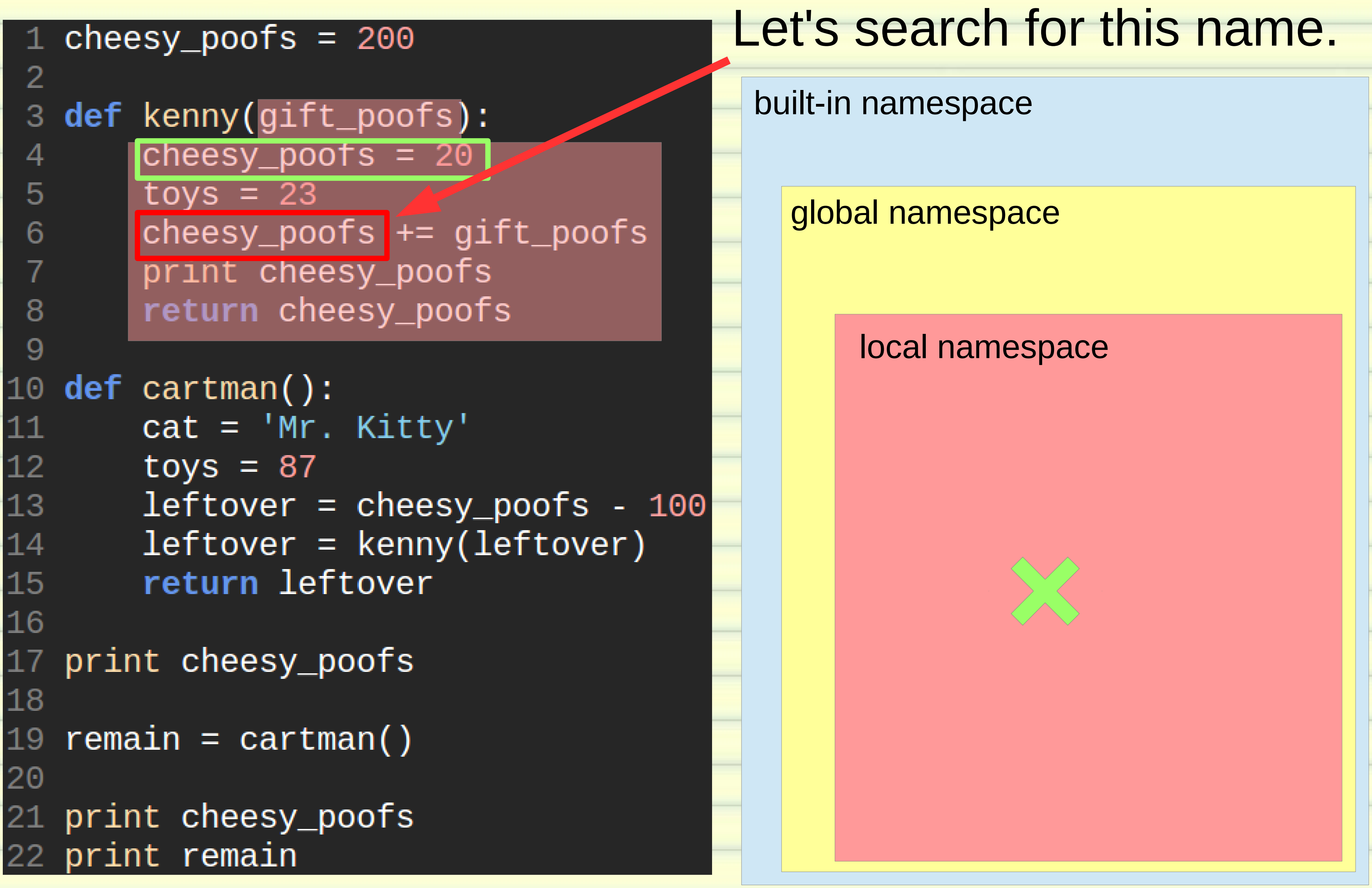


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.



Namespaces & Variable Scope



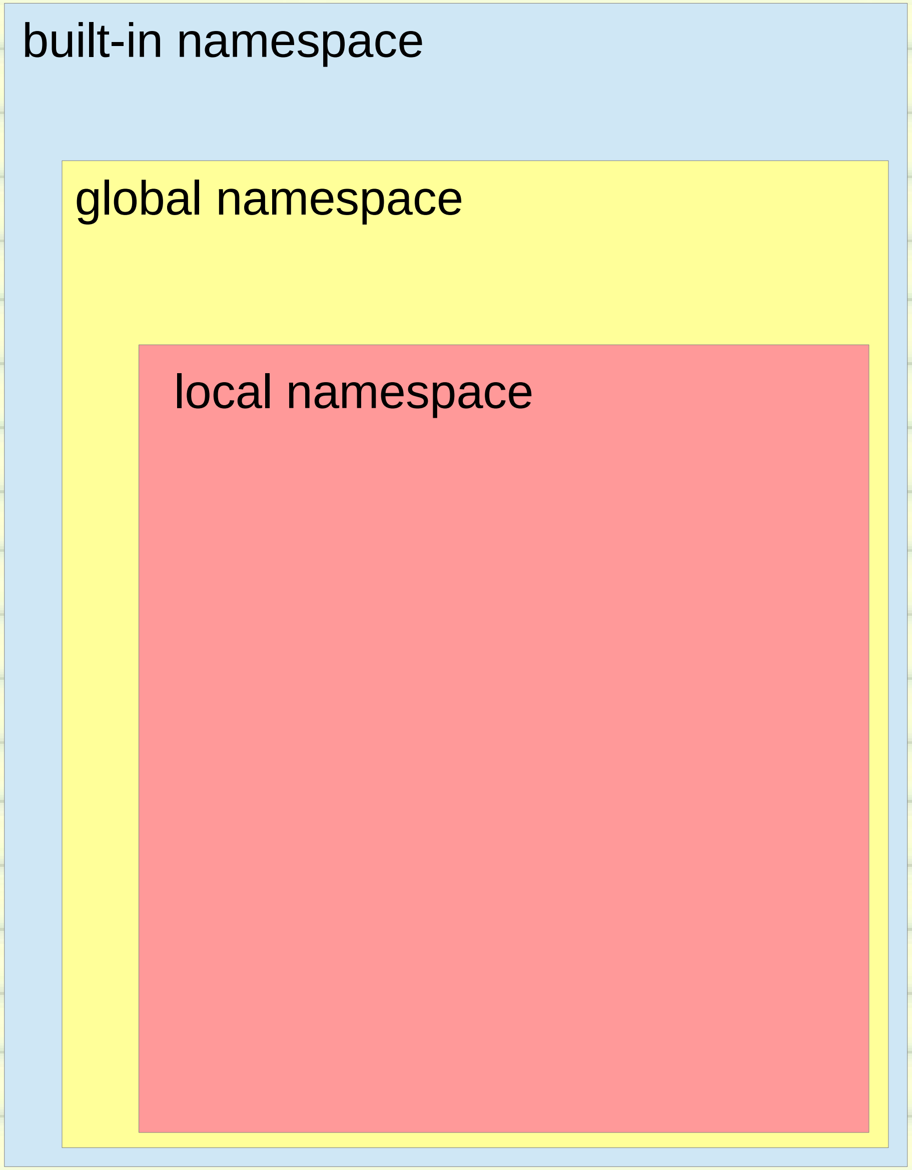
```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

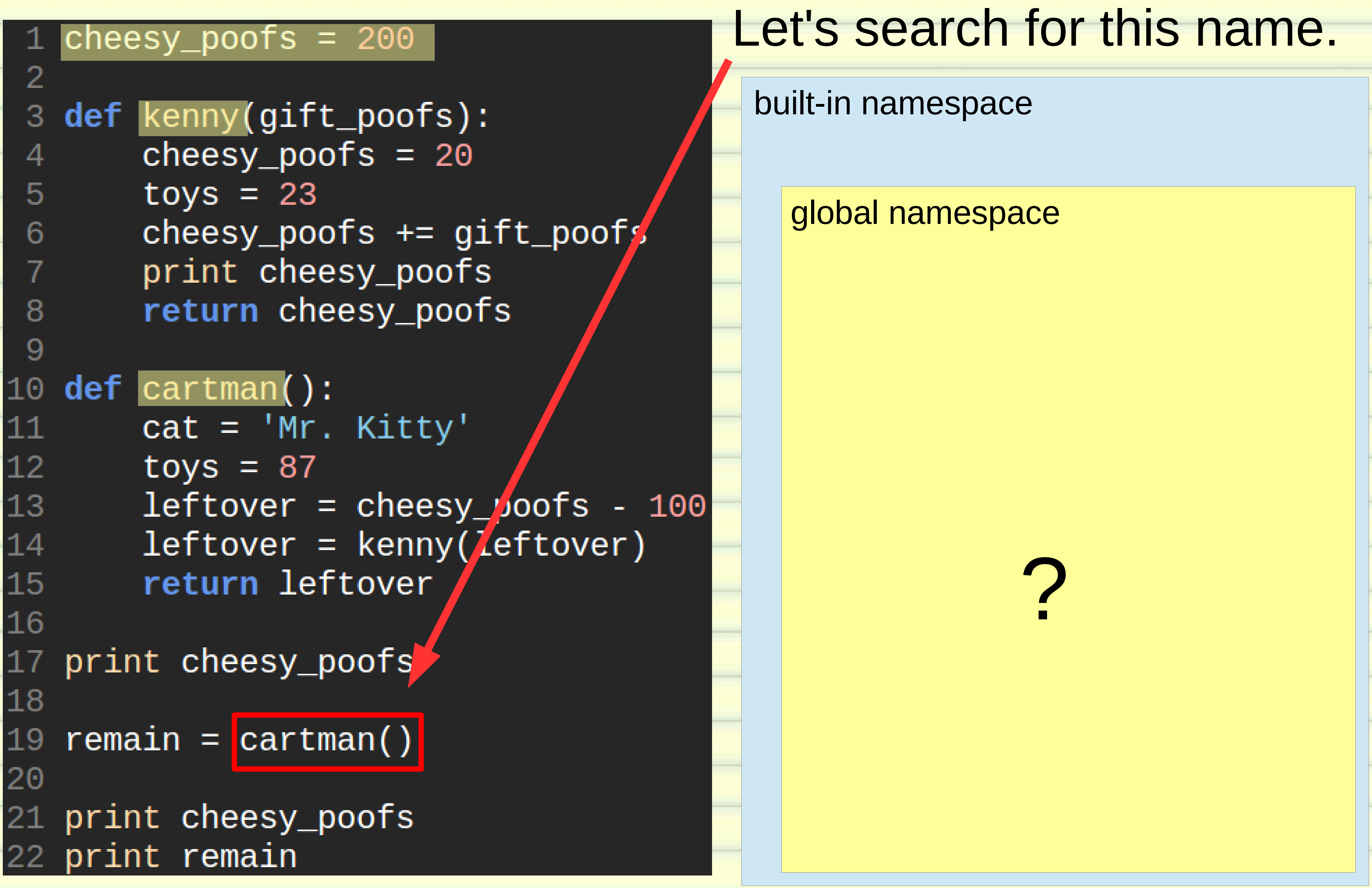
Namespaces & Variable Scope

```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.



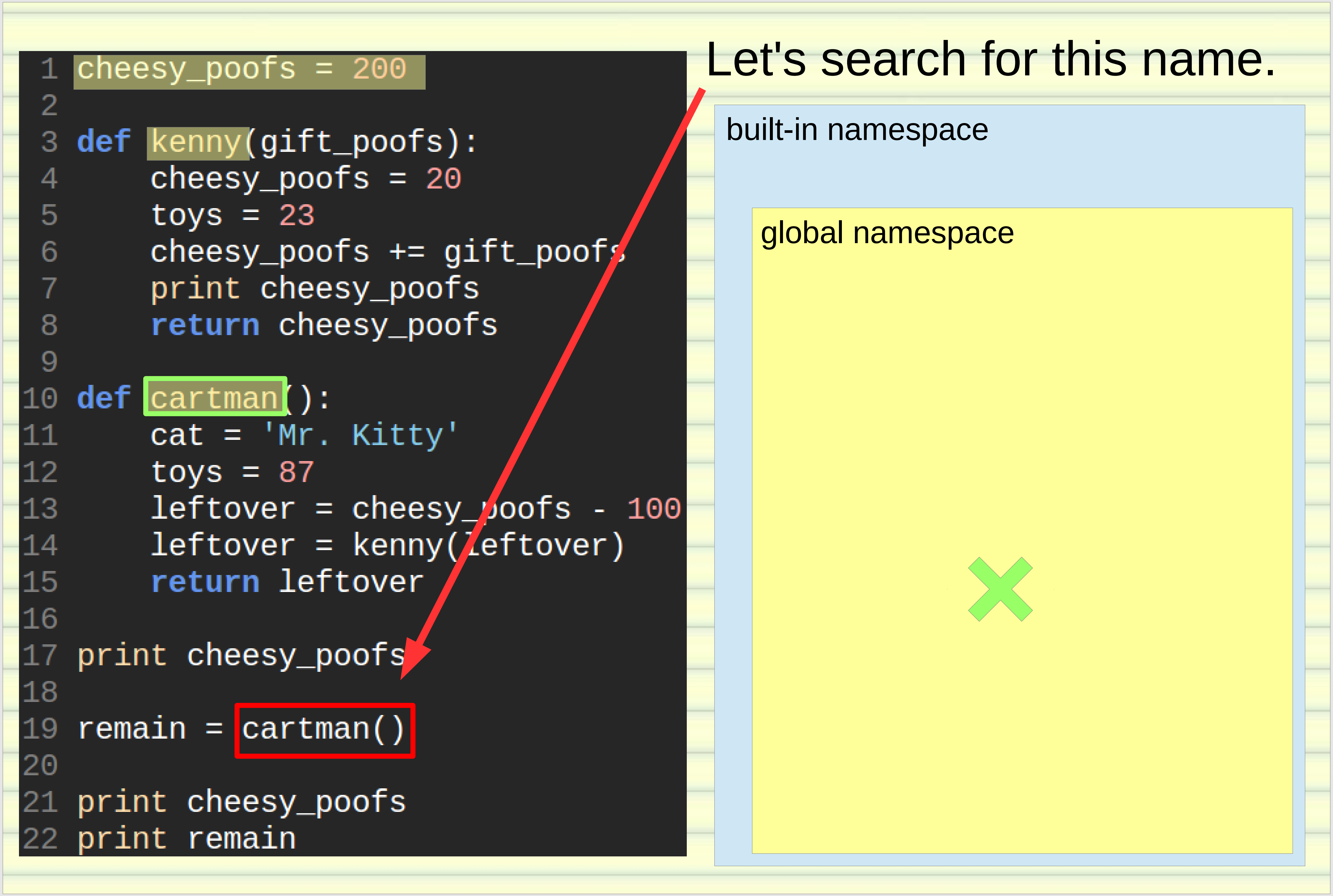
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

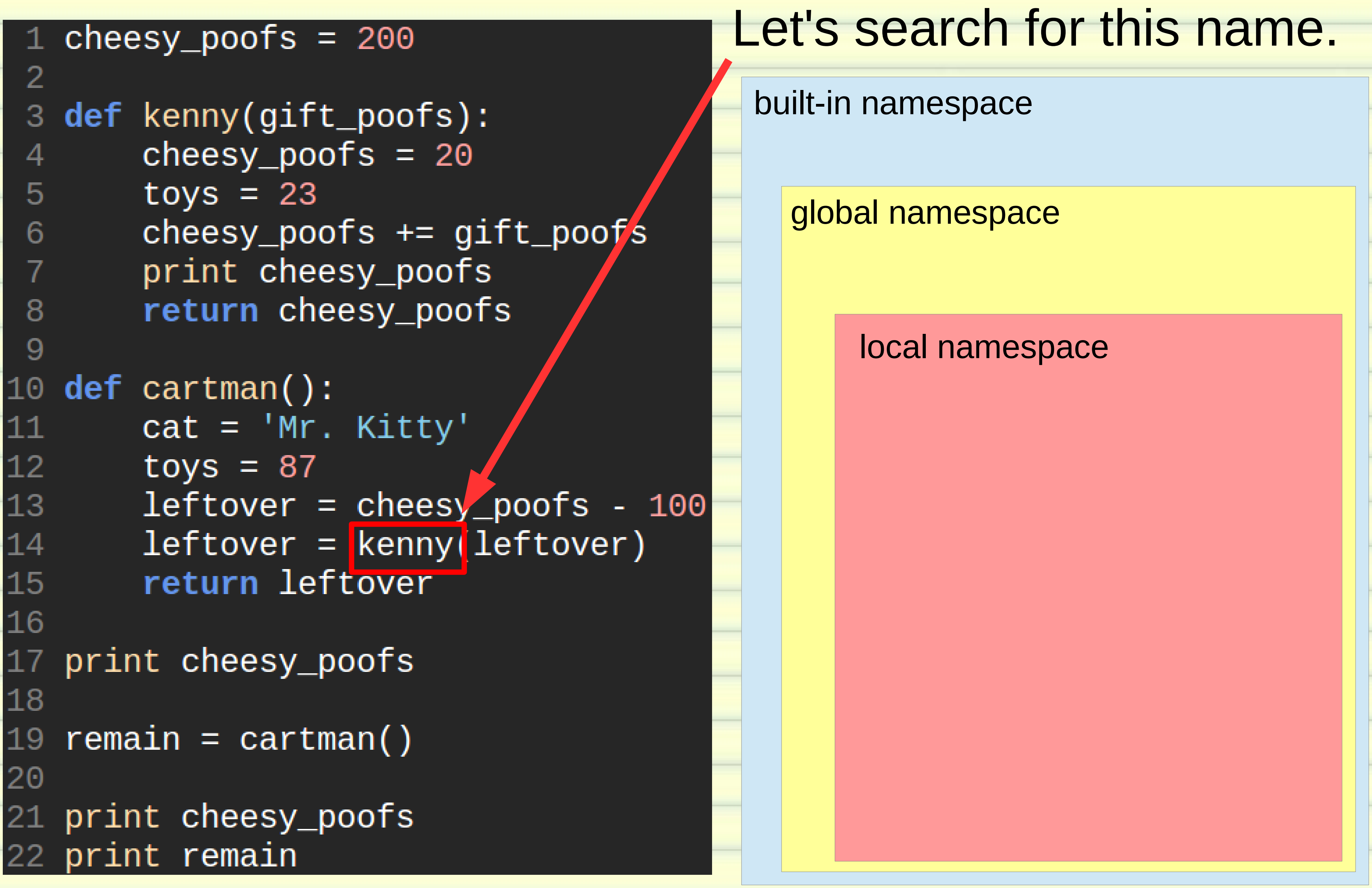
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

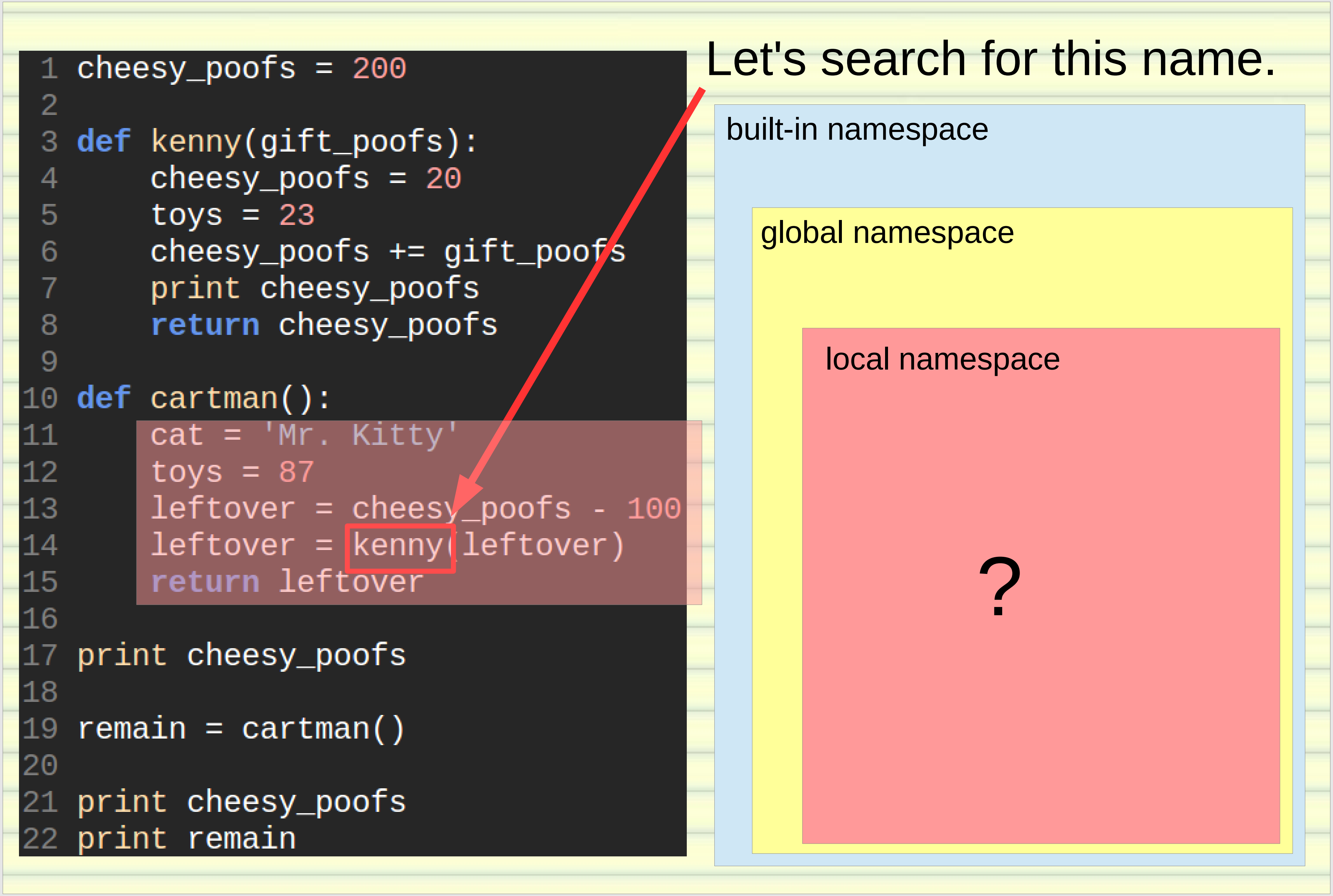
Let's search for this name.

built-in namespace

global namespace

local namespace

Namespaces & Variable Scope



```python
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.

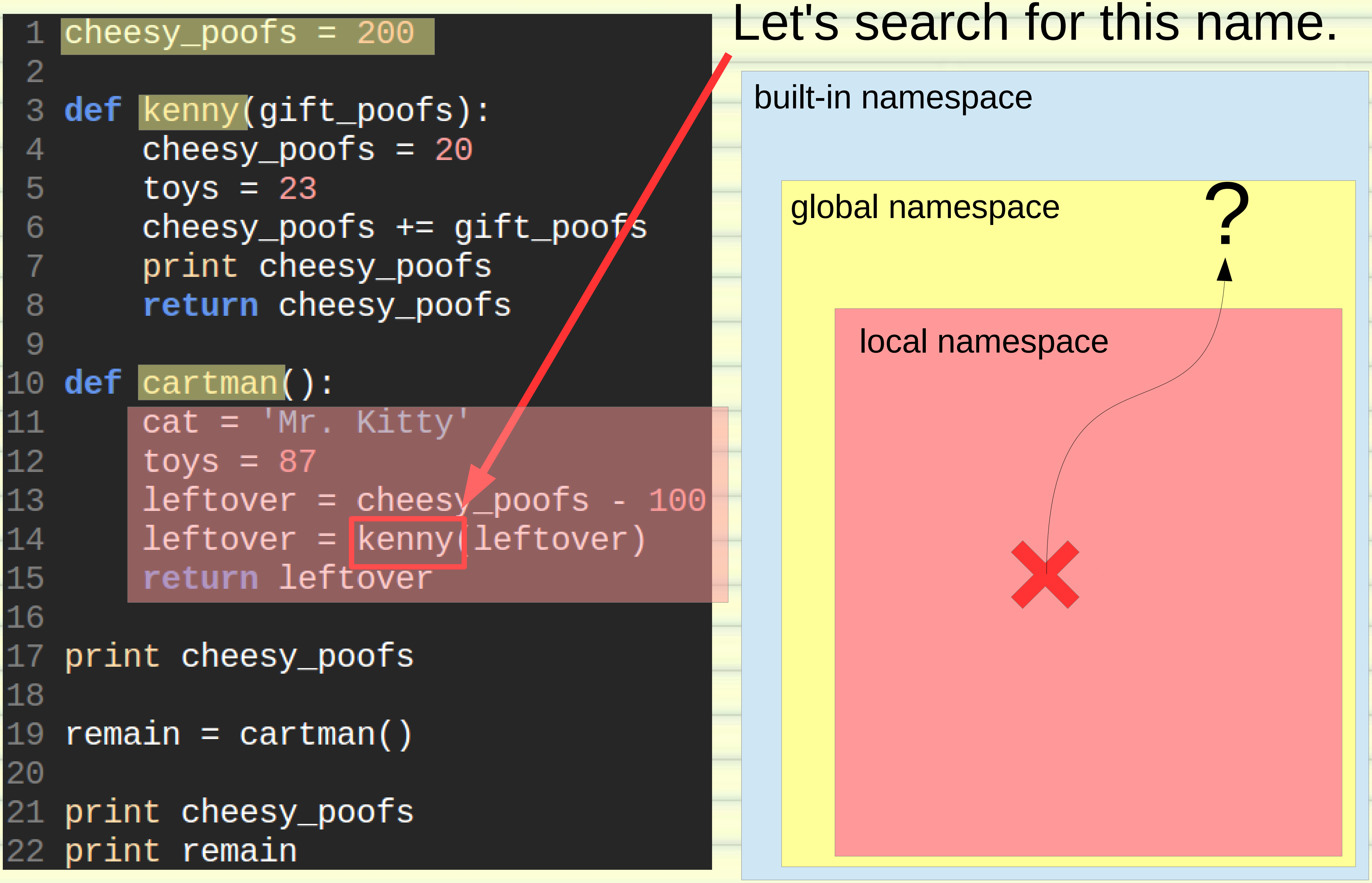
built-in namespace

global namespace

local namespace

?

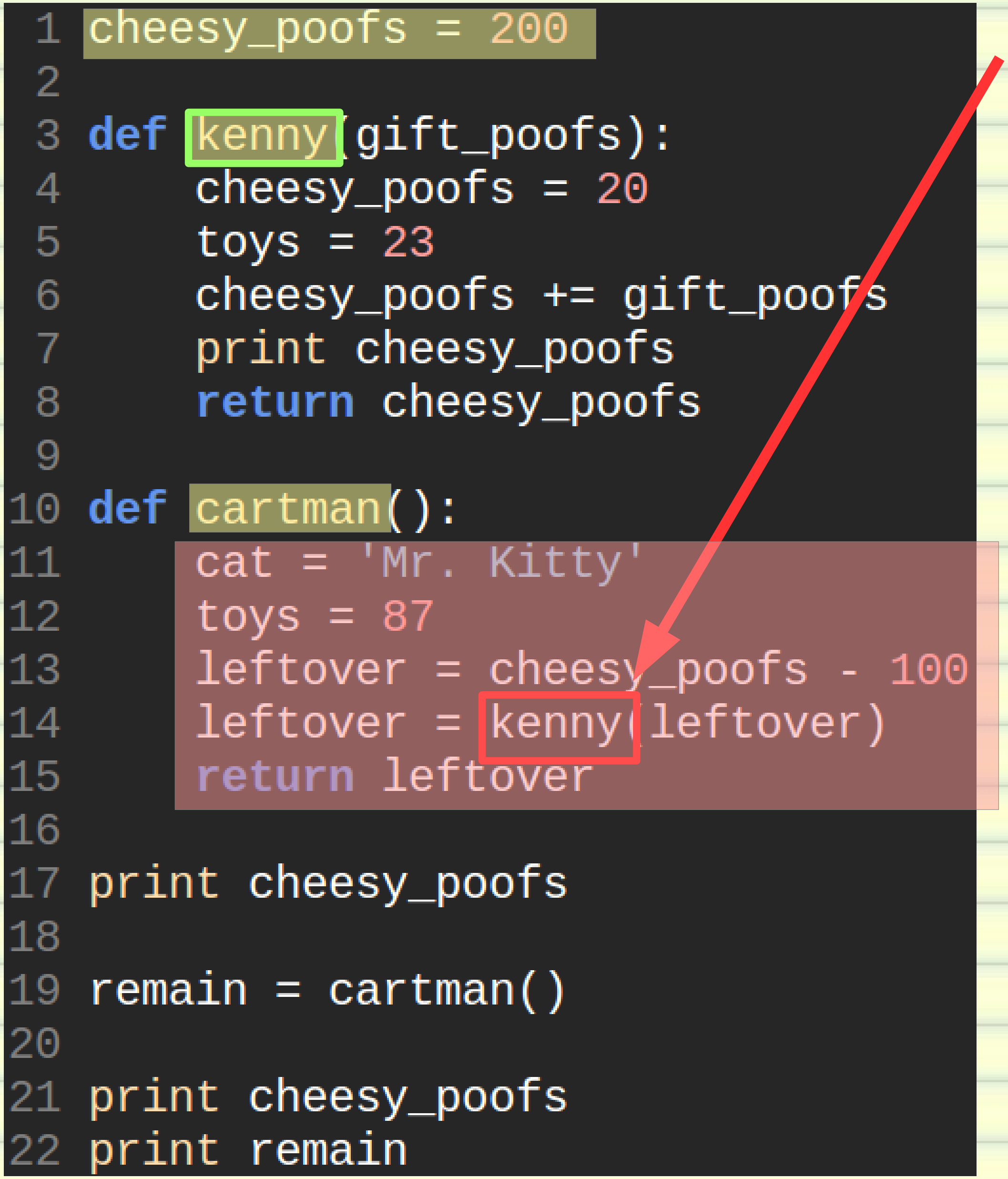
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

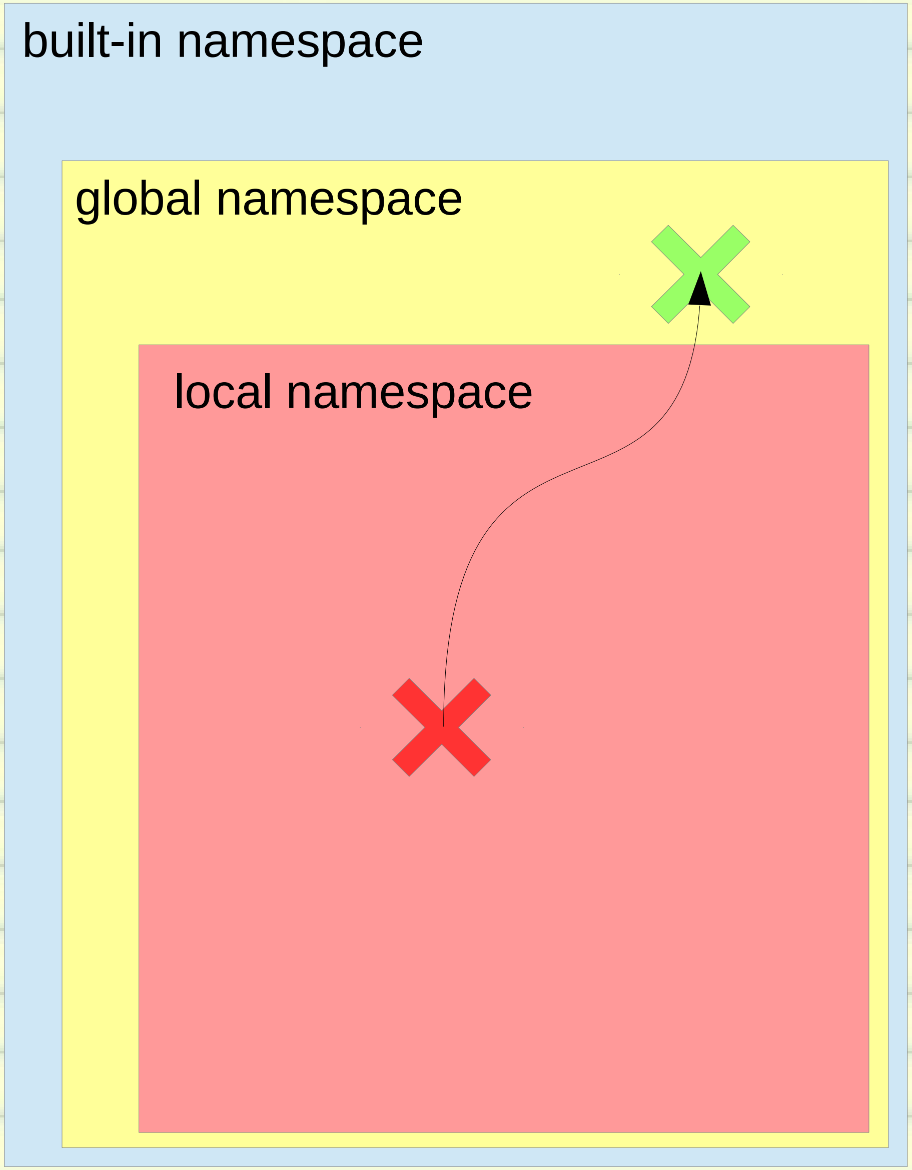
Let's search for this name.

Namespaces & Variable Scope

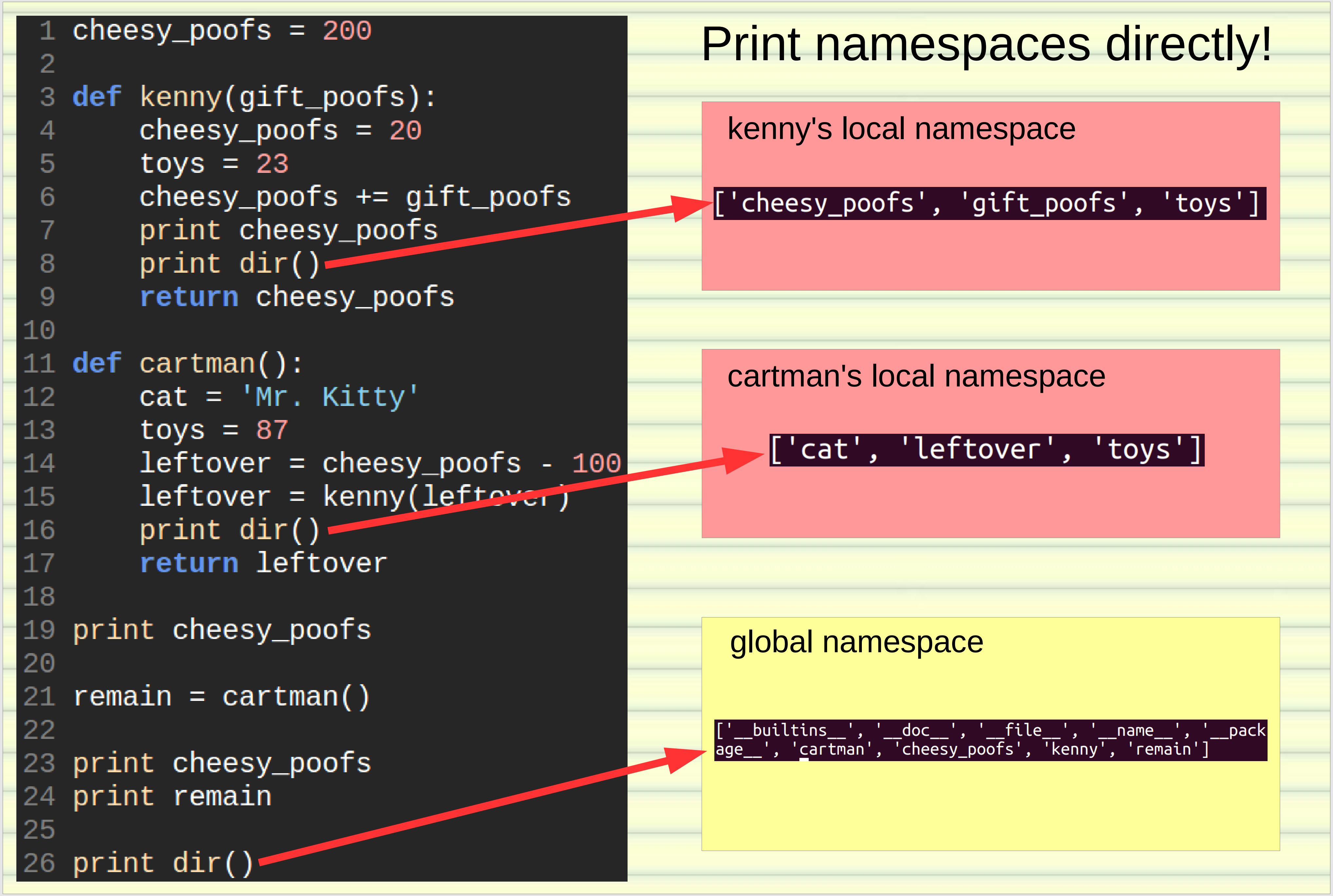


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's search for this name.



Namespaces & Variable Scope



```python
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

Print namespaces directly!

kenny's local namespace

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

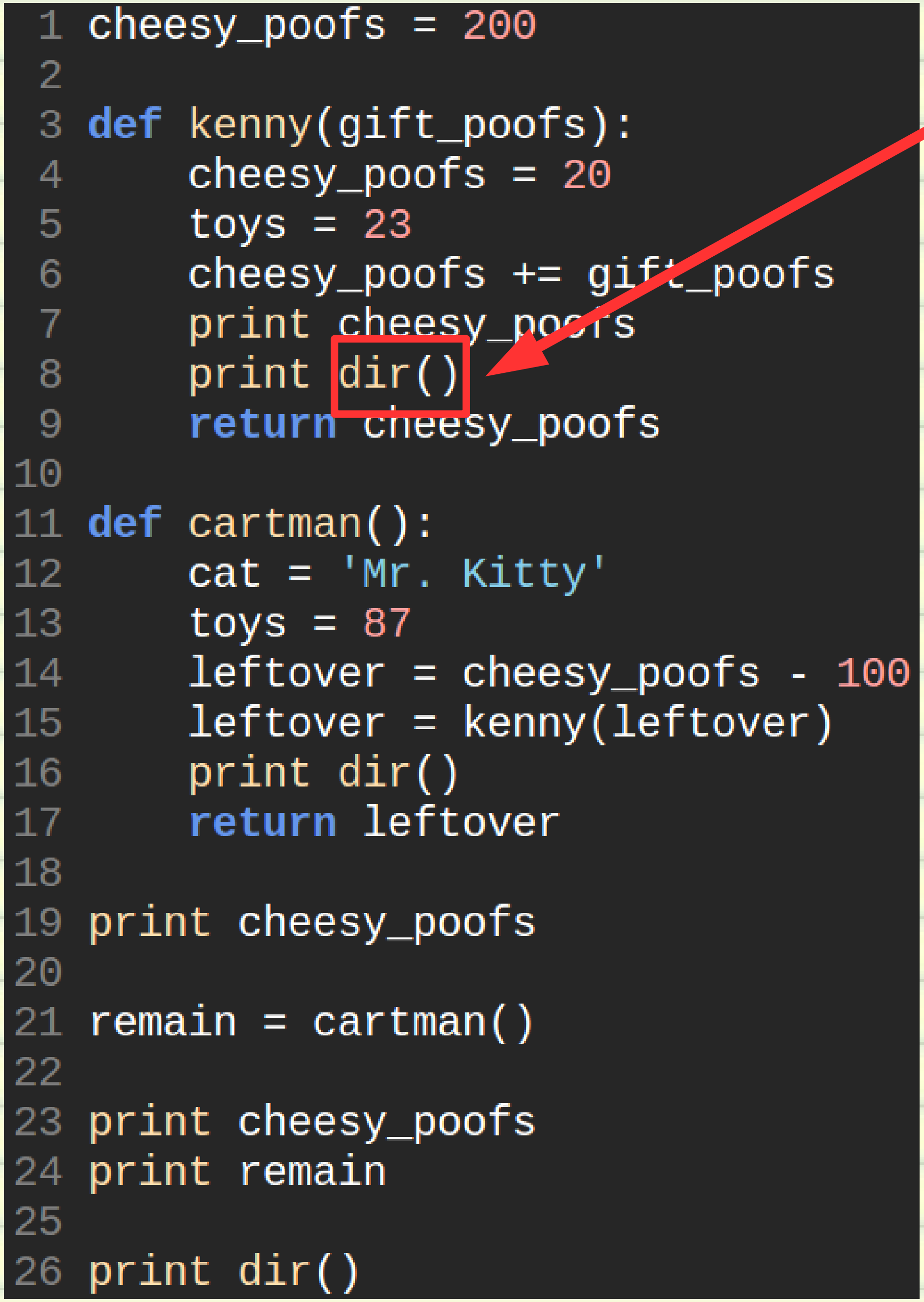
cartman's local namespace

```
['cat', 'leftover', 'toys']
```

global namespace

```
['__builtins__', '__doc__', '__file__', '__name__', '__pack
age__', 'cartman', 'cheesy_poofs', 'kenny', 'remain']
```

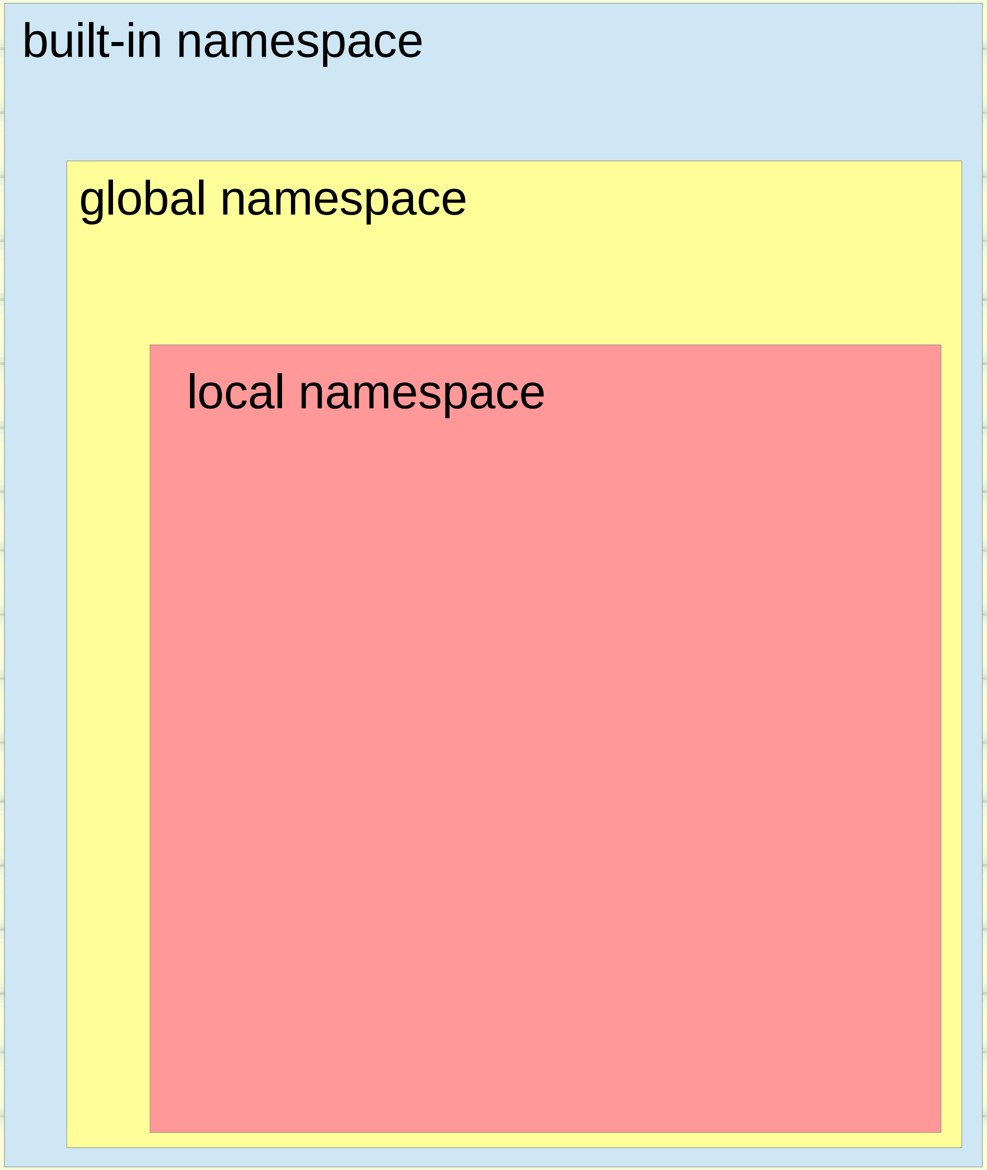
Namespaces & Variable Scope



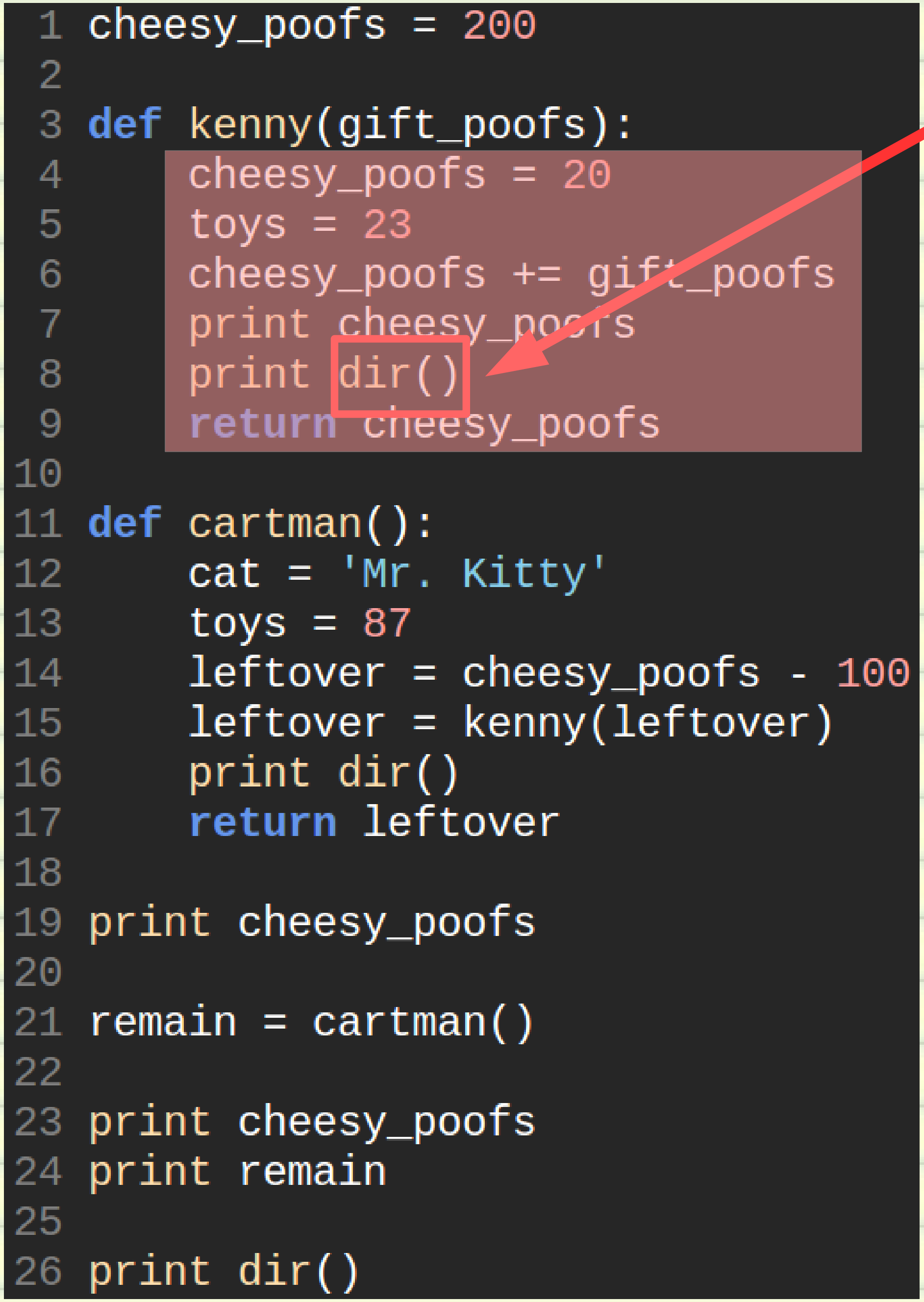
```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

btw...

Let's search for this name.



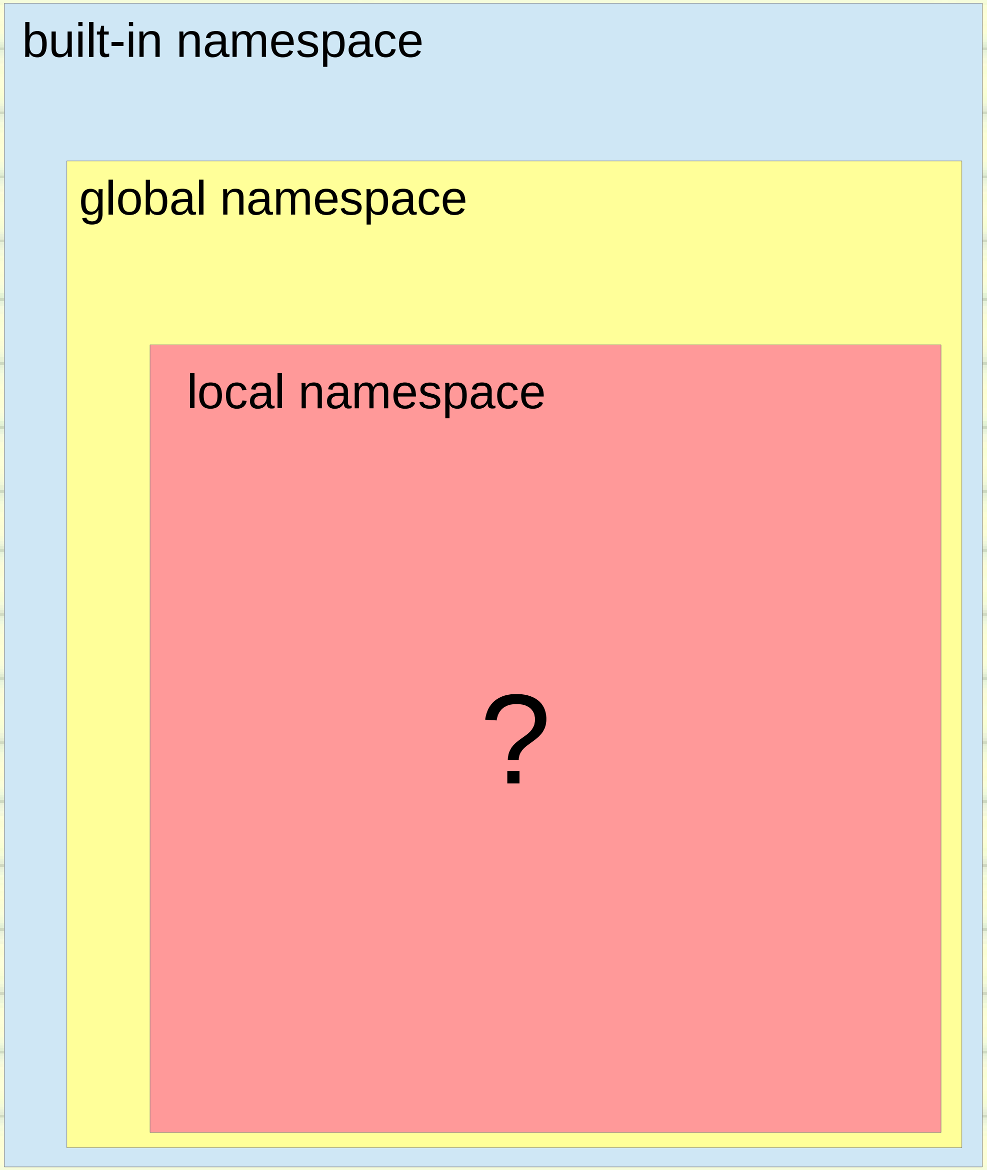
Namespaces & Variable Scope



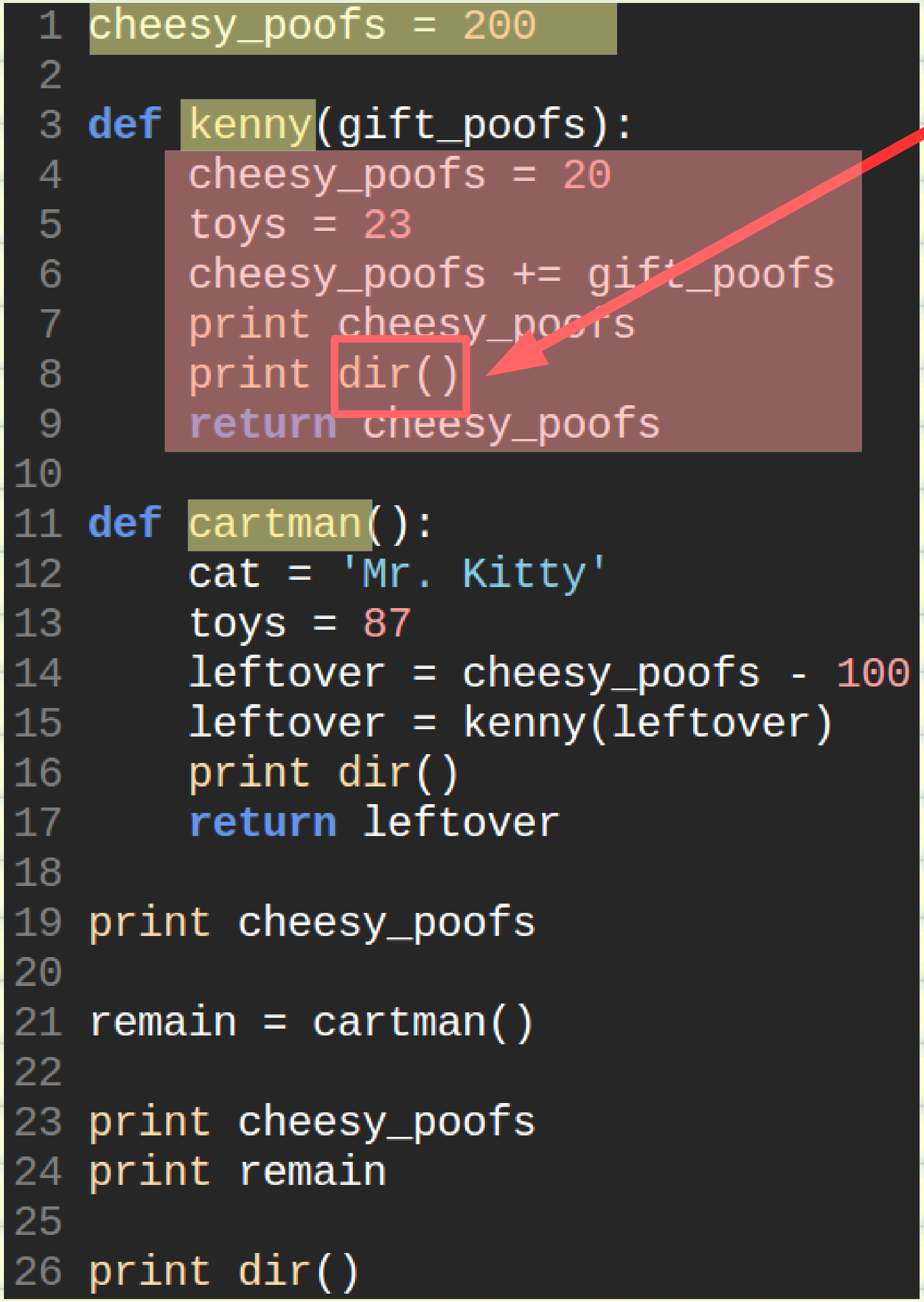
```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

btw...

Let's search for this name.



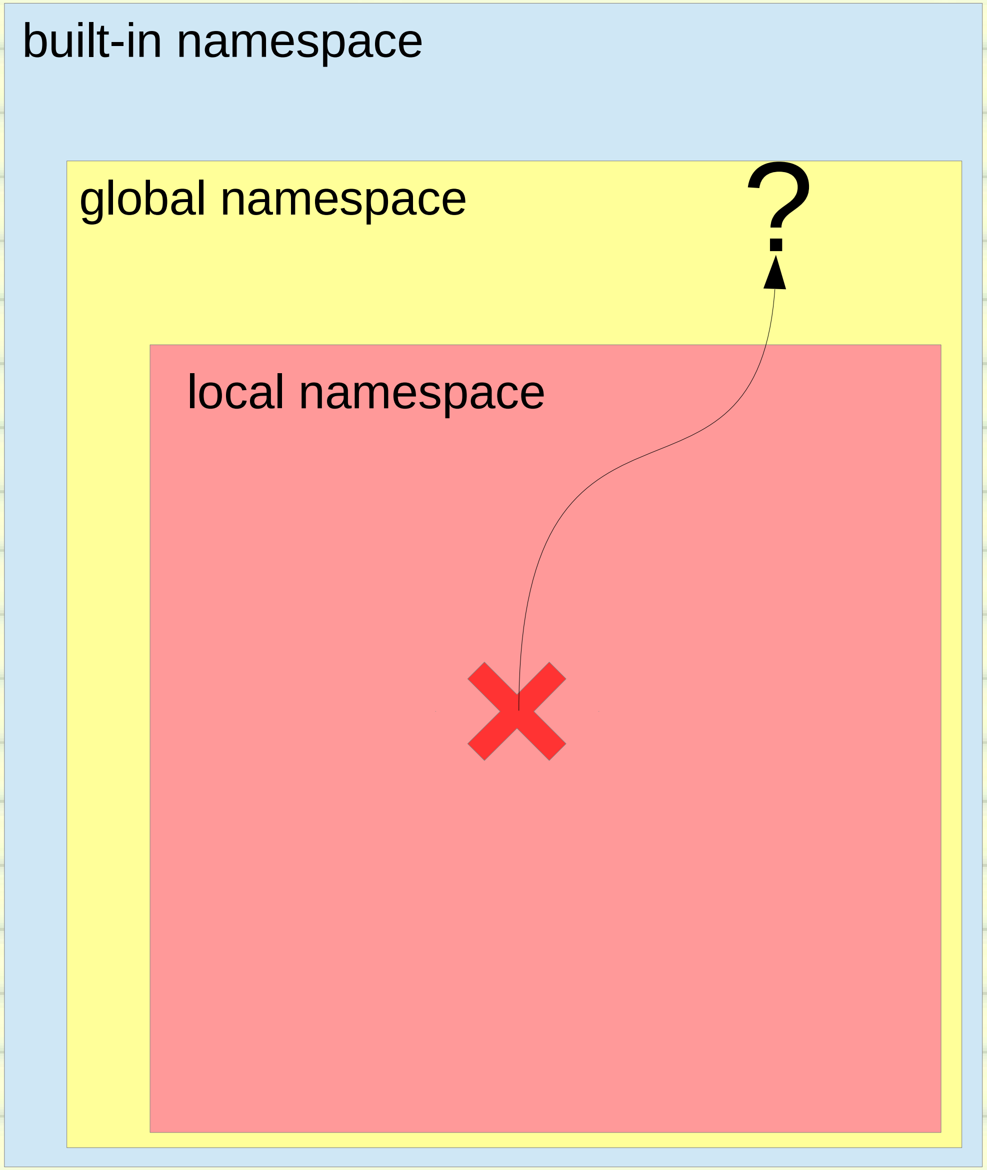
Namespaces & Variable Scope



```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

btw...

Let's search for this name.



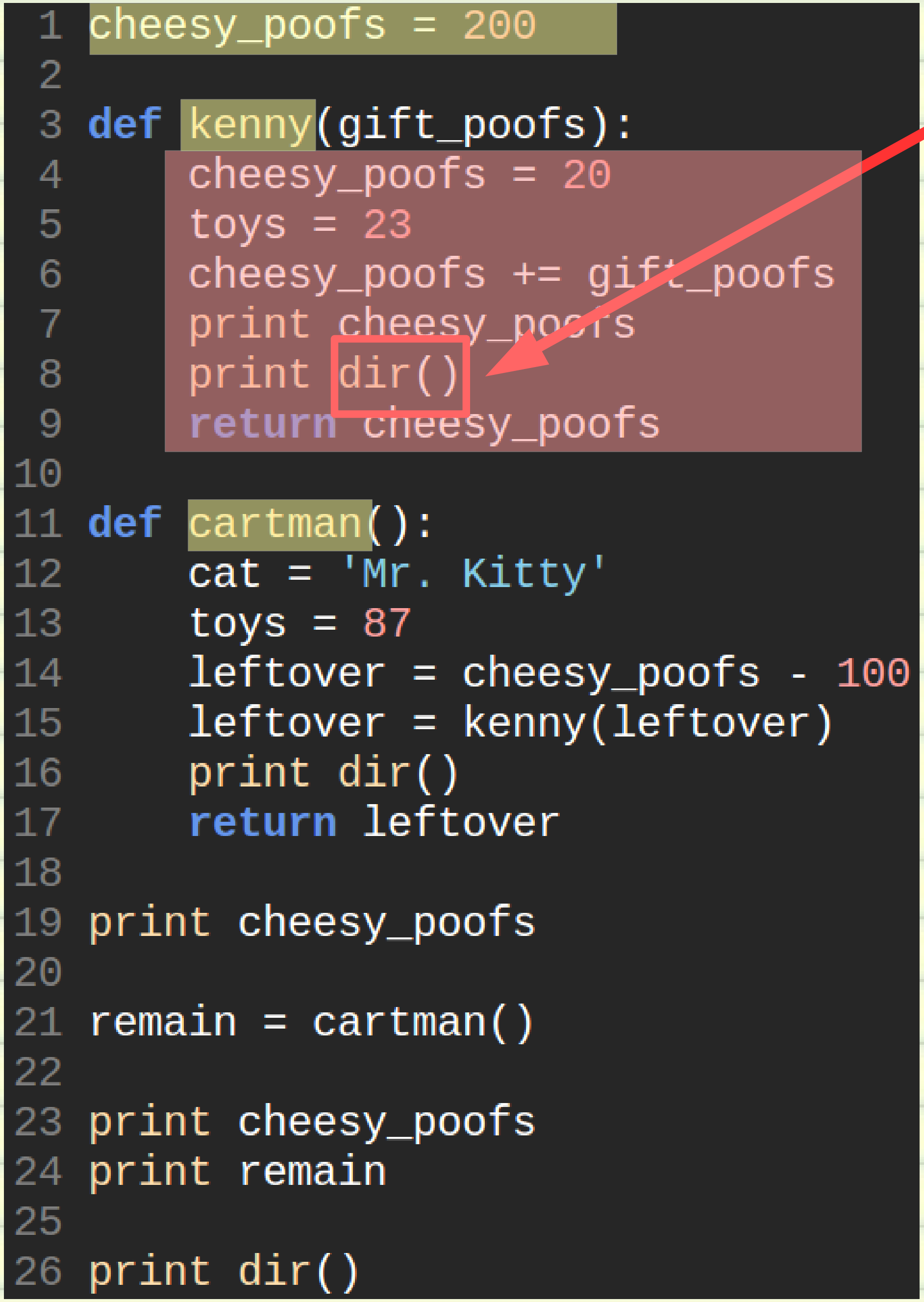
built-in namespace

global namespace

?

local namespace

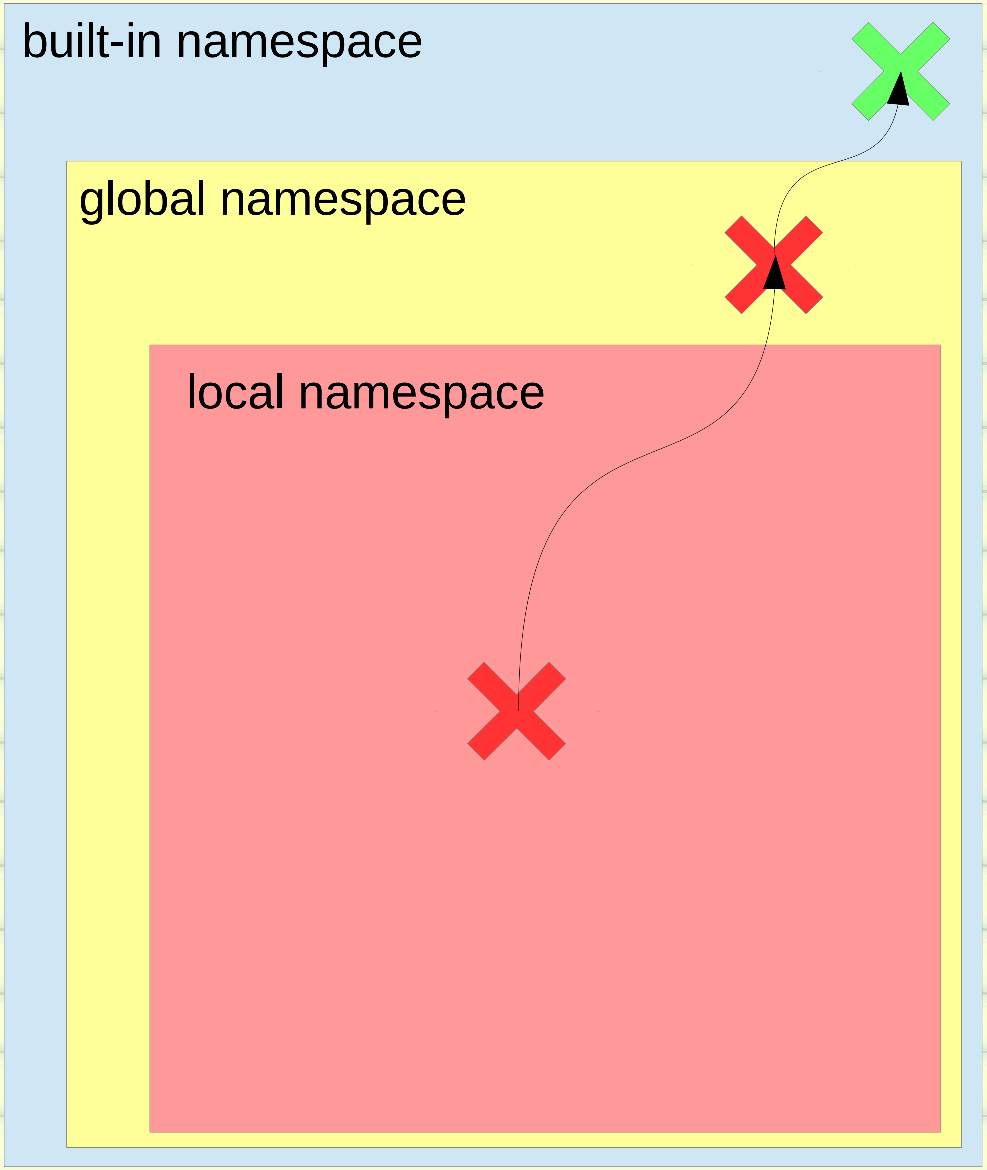
Namespaces & Variable Scope



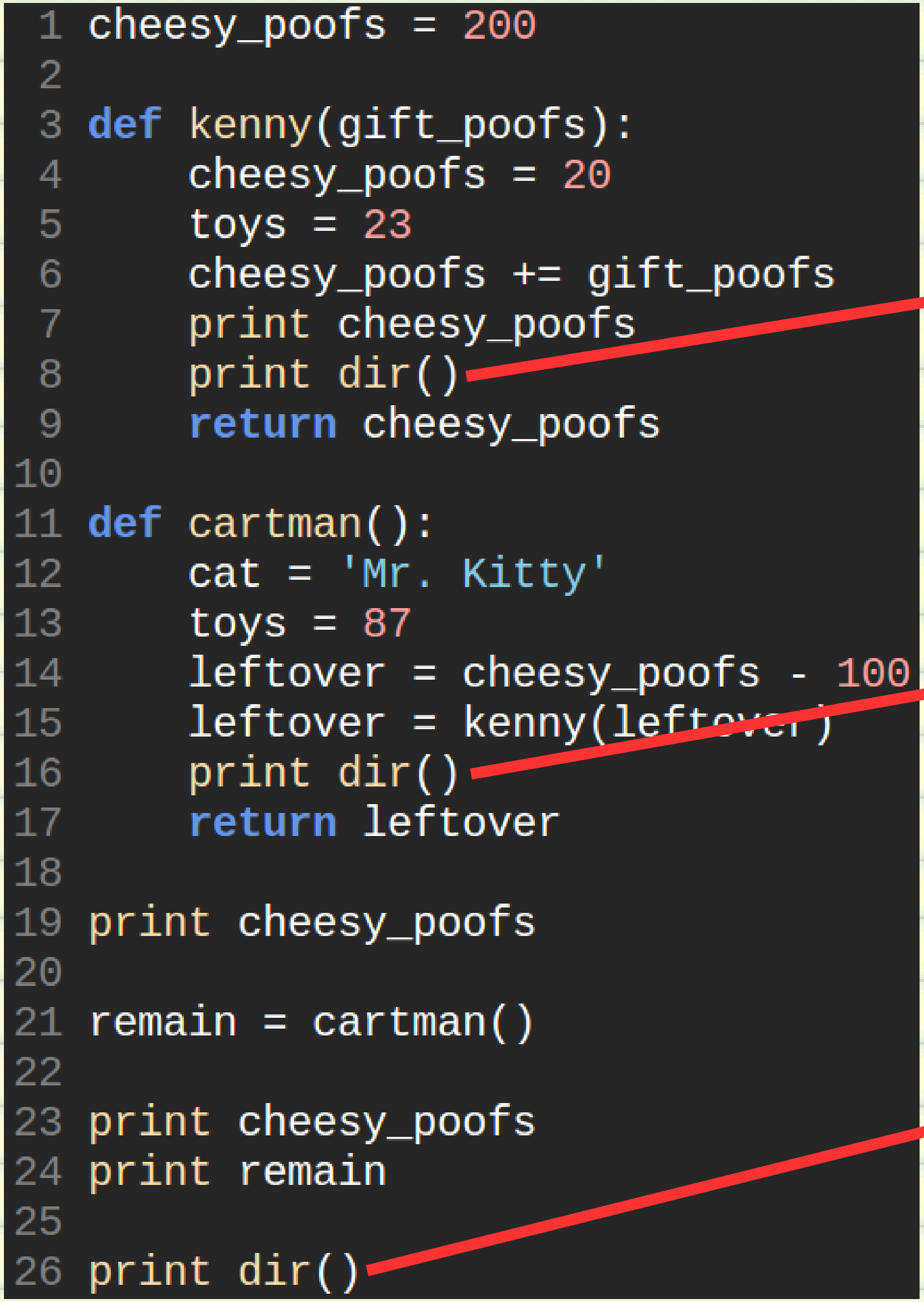
```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

btw...

Let's search for this name.



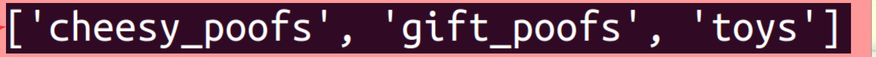
Namespaces & Variable Scope



```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    print dir()
    return cheesy_poofs

def cartman():
    cat = 'Mr. Kitty'
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    print dir()
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain

print dir()
```

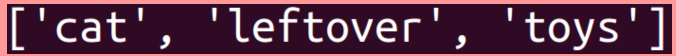
Print namespaces directly!

kenny's local namespace



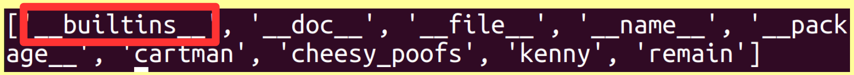
```
['cheesy_poofs', 'gift_poofs', 'toys']
```

cartman's local namespace



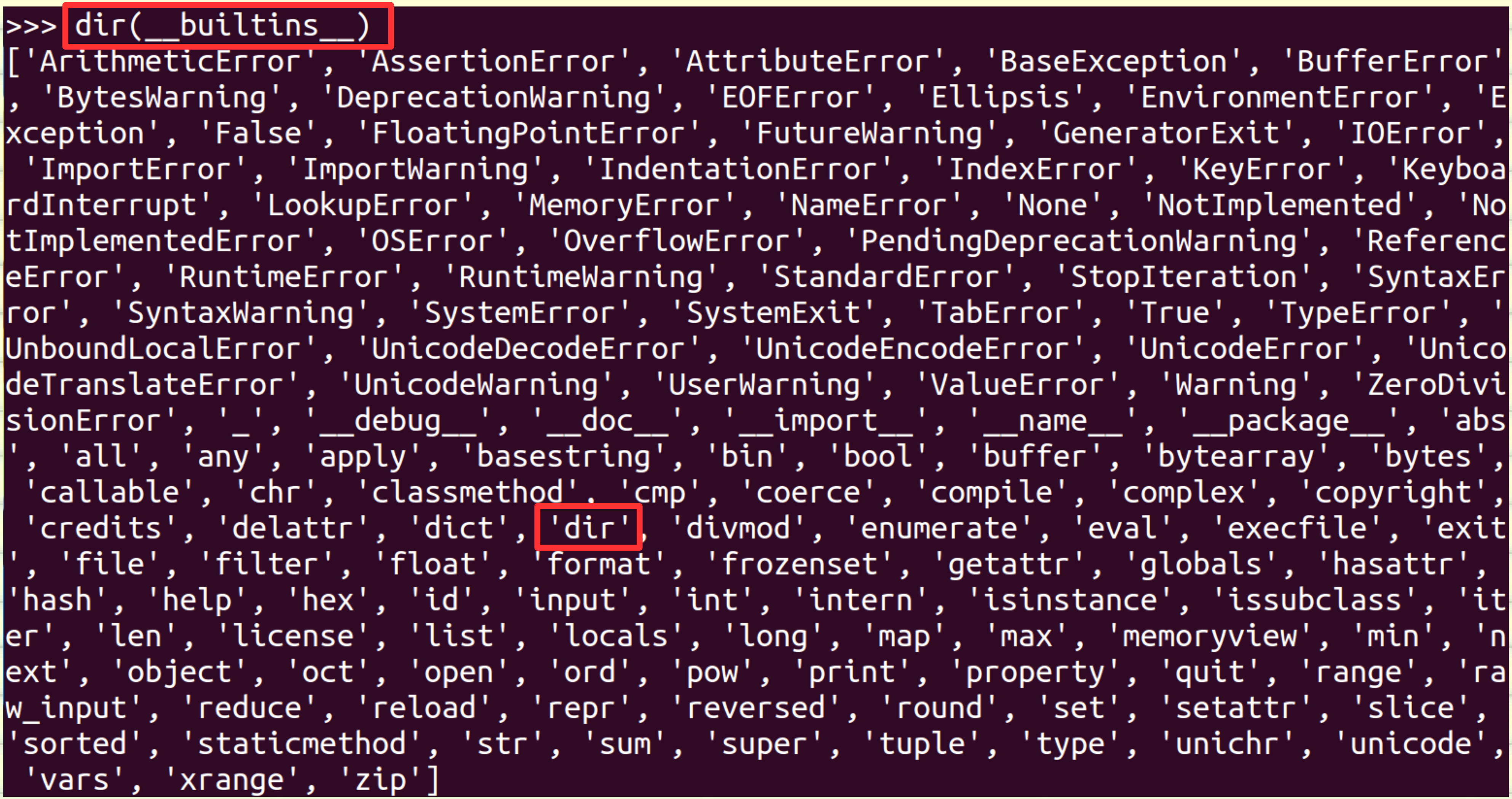
```
['cat', 'leftover', 'toys']
```

global namespace



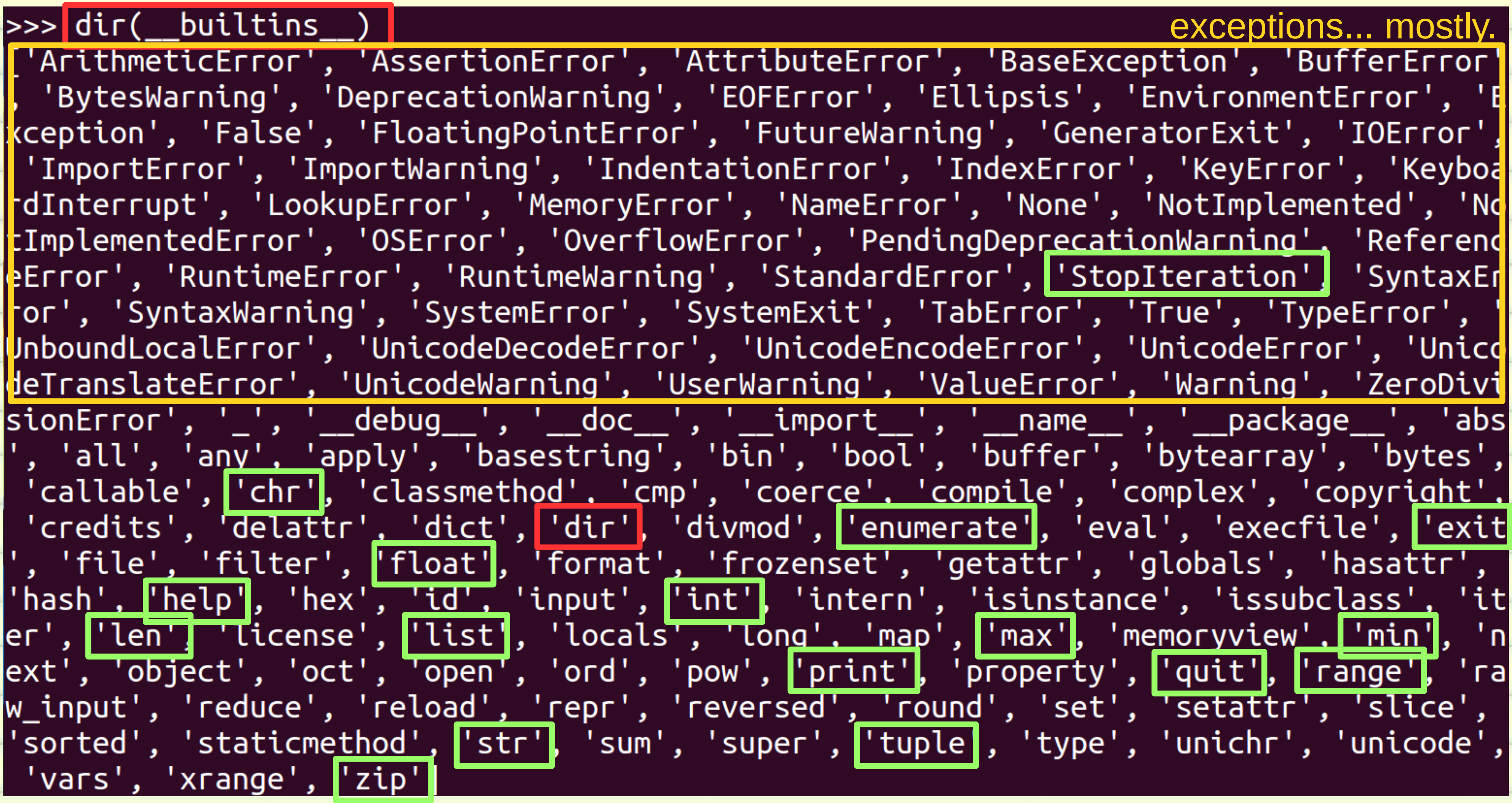
```
['__builtins__', '__doc__', '__file__', '__name__', '__pack
age__', 'cartman', 'cheesy_poofs', 'kenny', 'remain']
```

Namespaces & Variable Scope



```
>>> 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', 'execfile', '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']
```

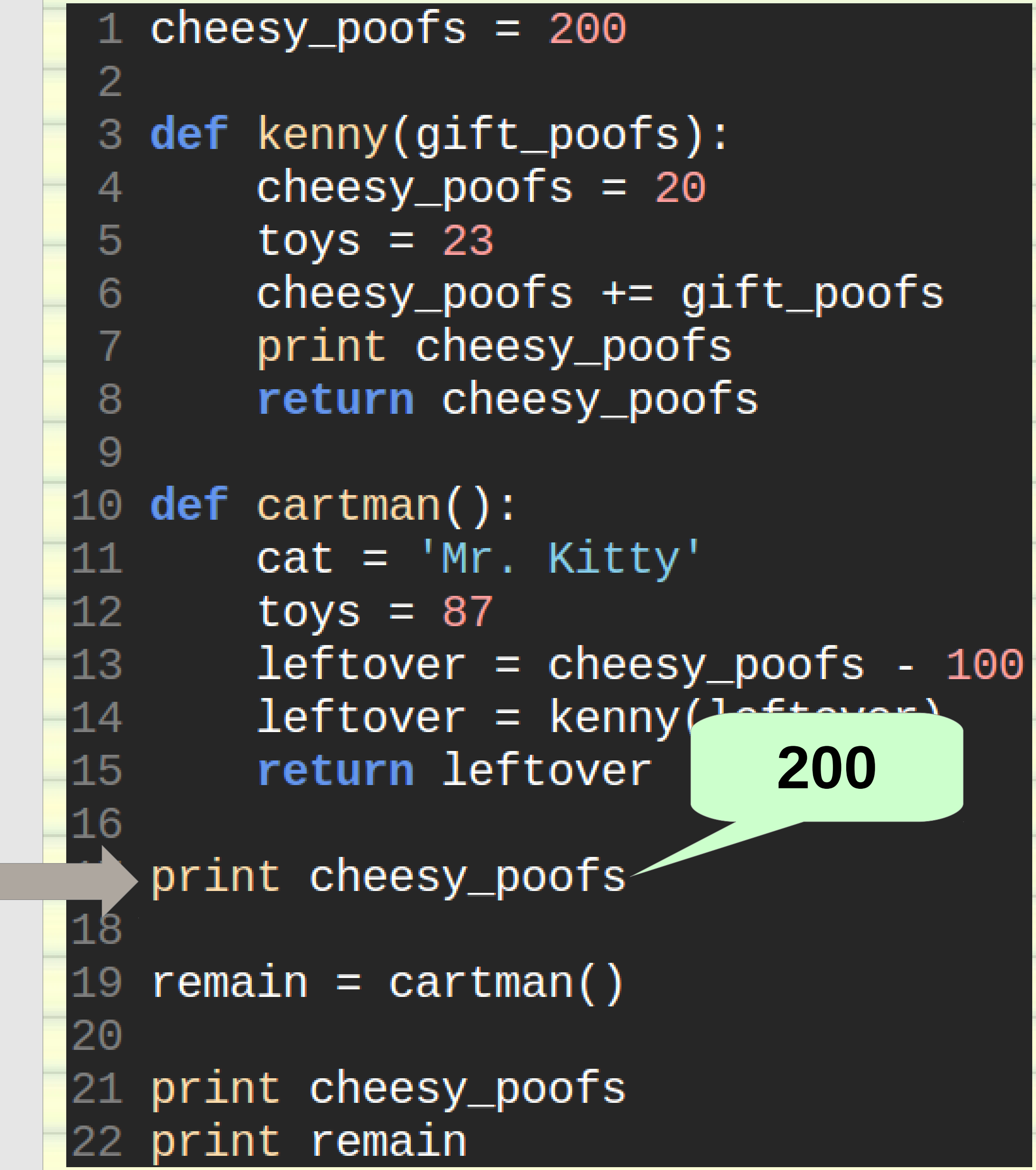
Namespaces & Variable Scope



exceptions... mostly.

```
>>> 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', 'execfile', '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']
```

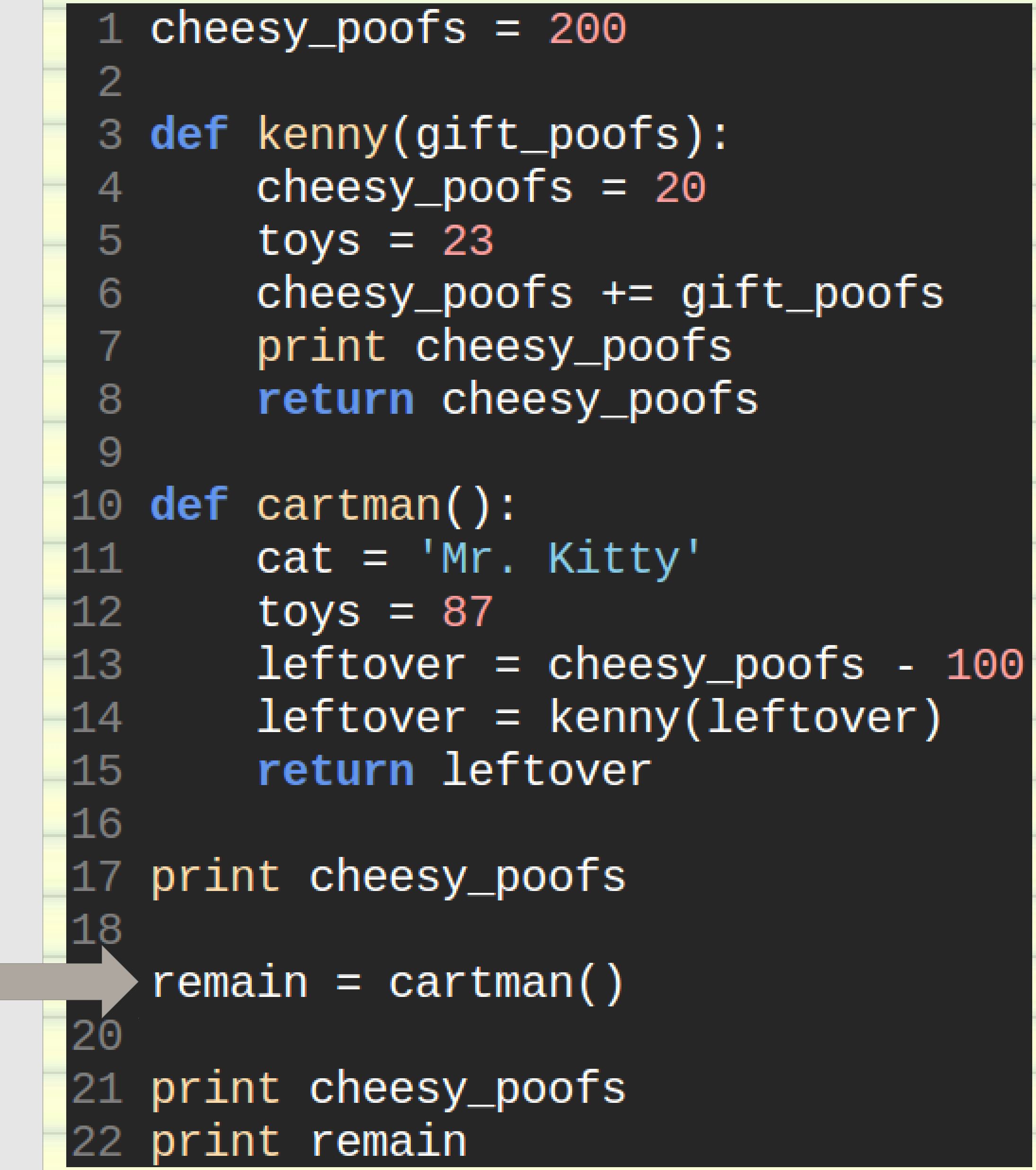
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

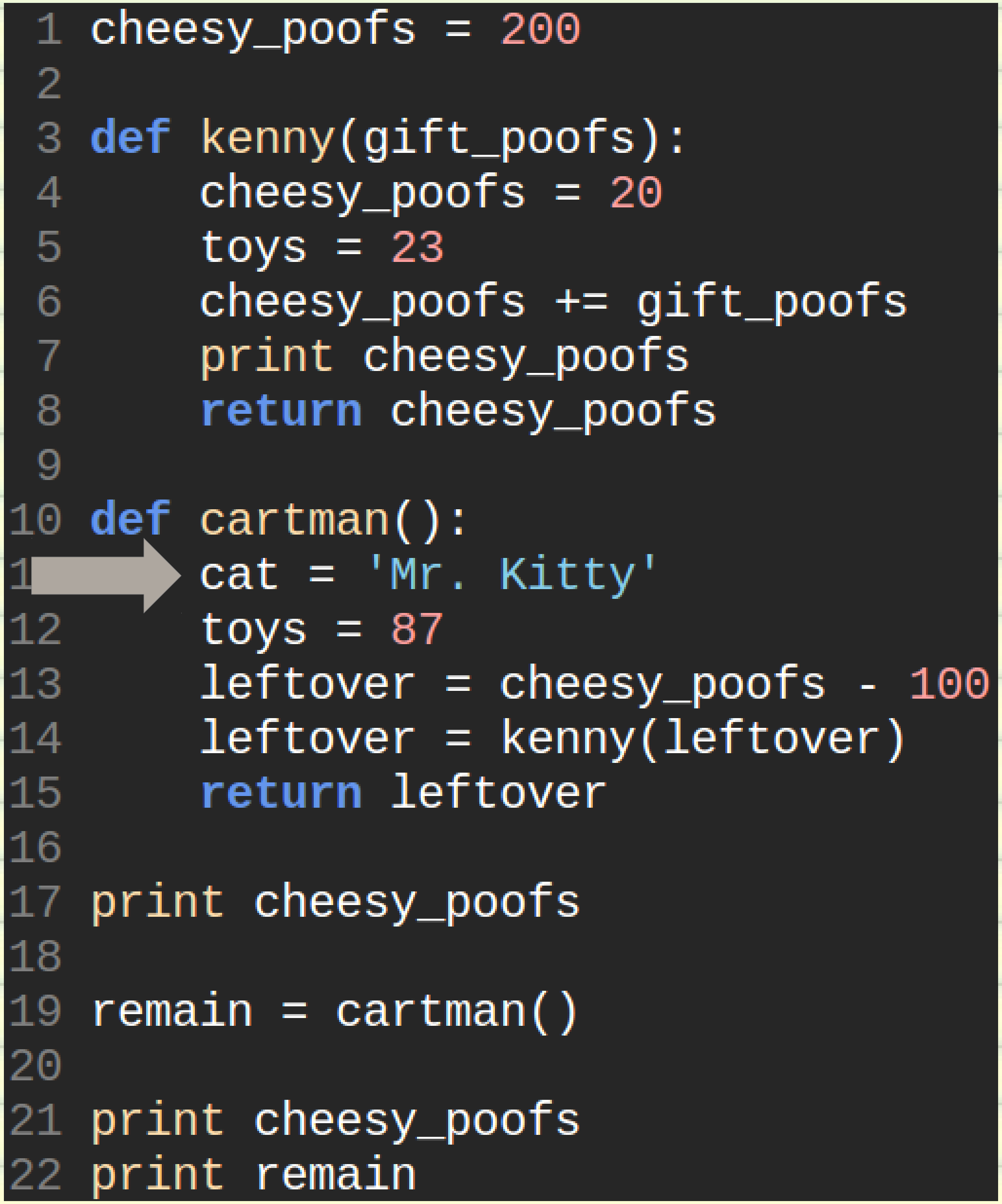
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

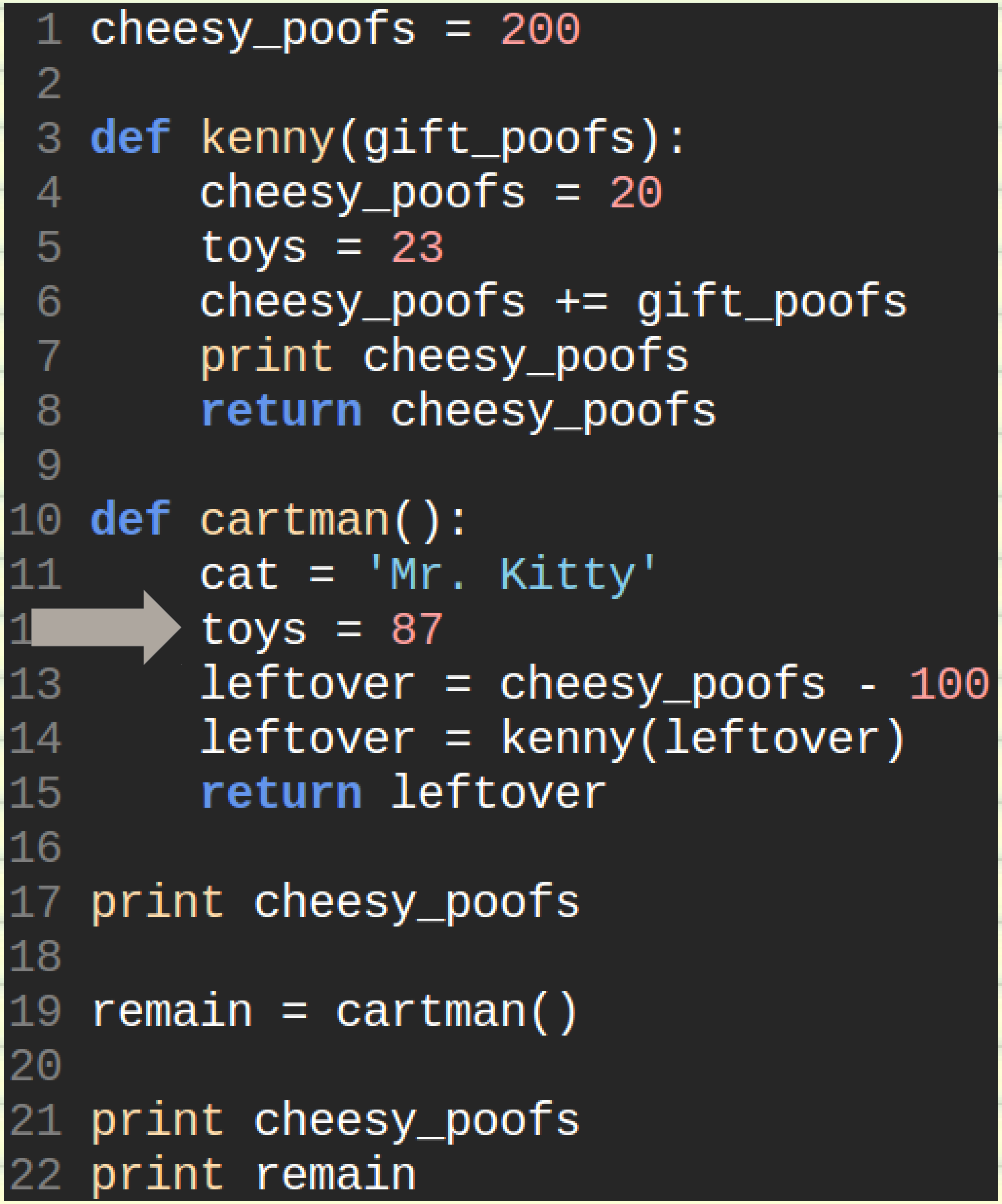
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

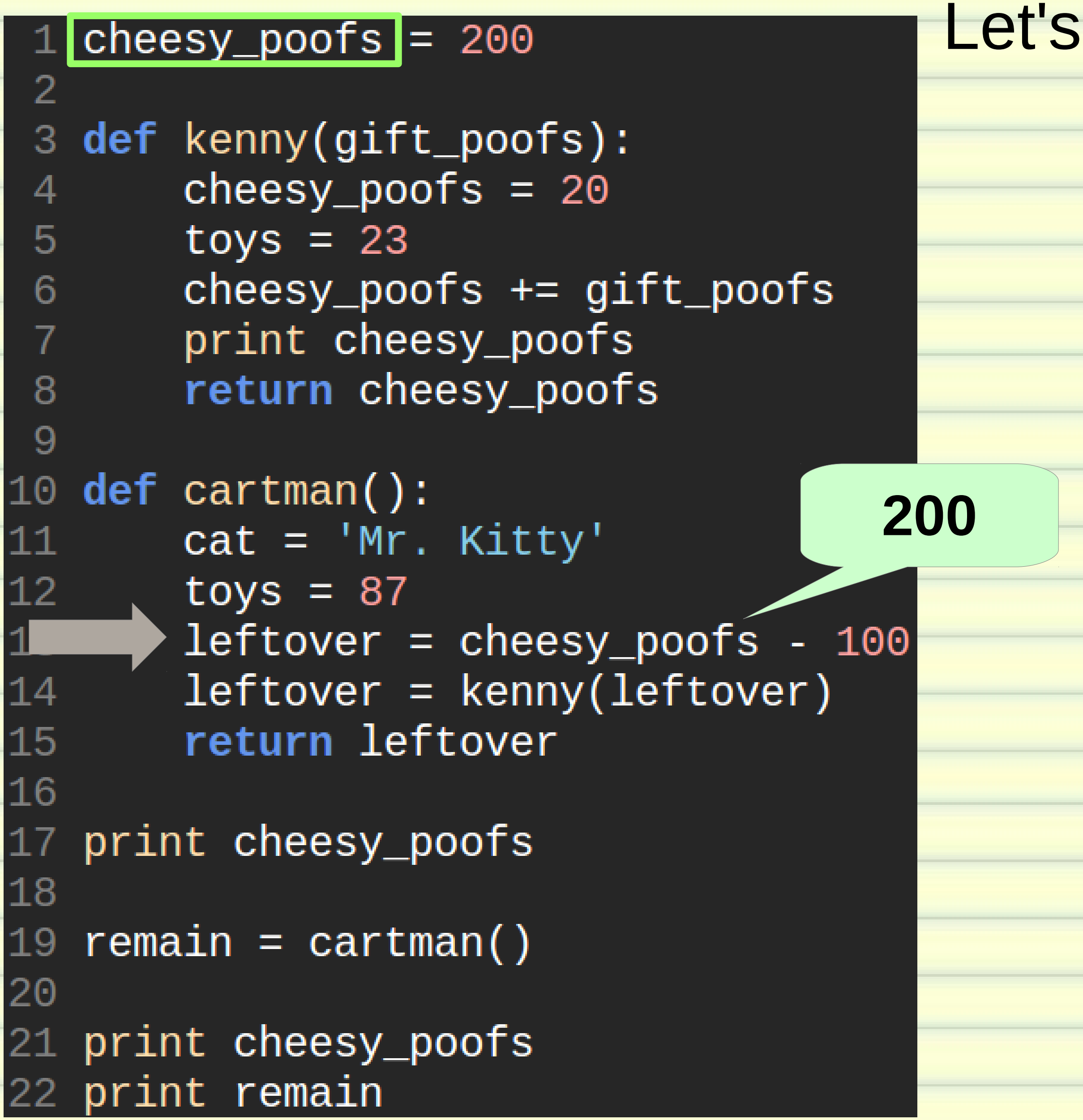
Namespaces & Variable Scope



```python
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

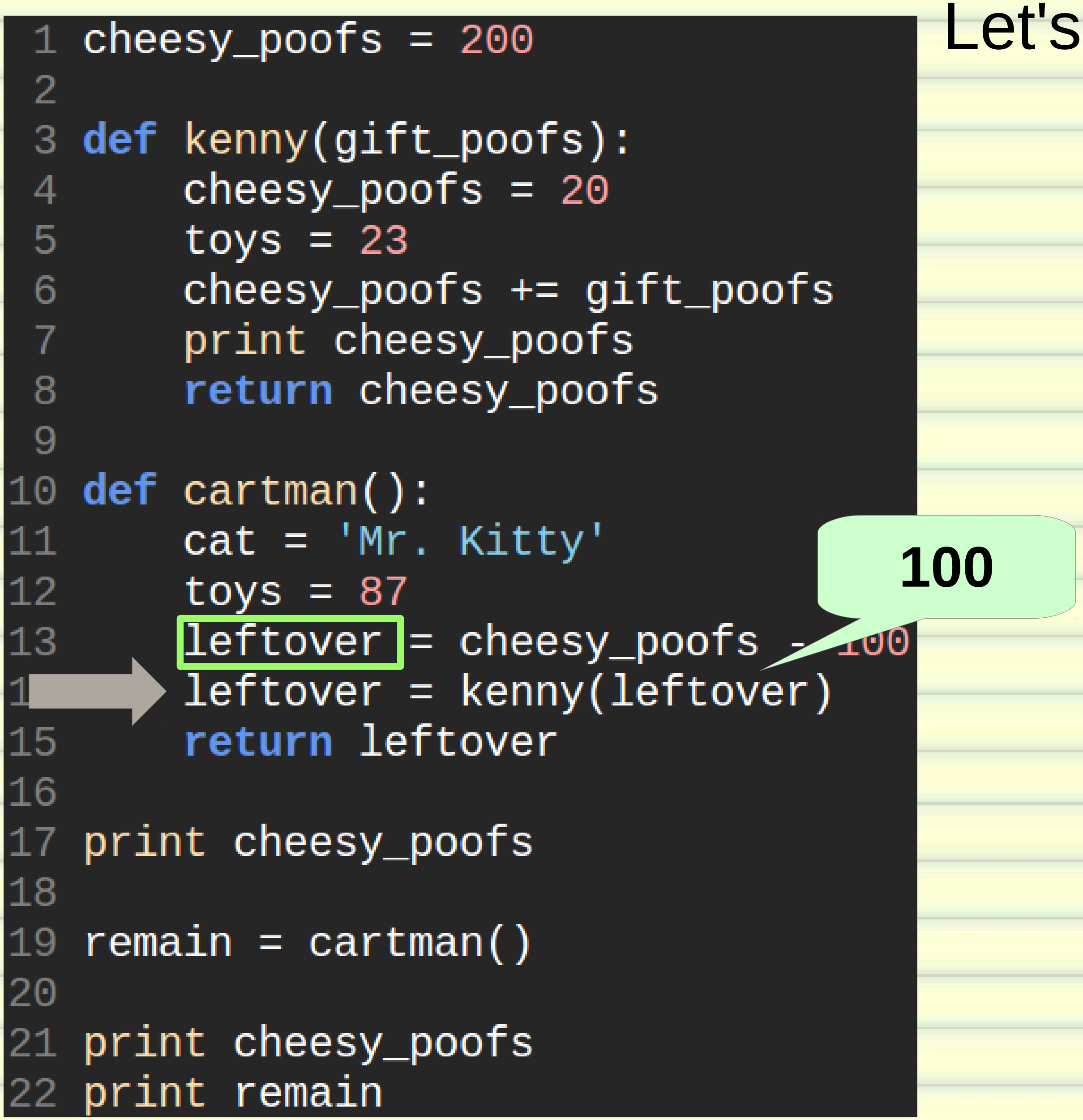
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

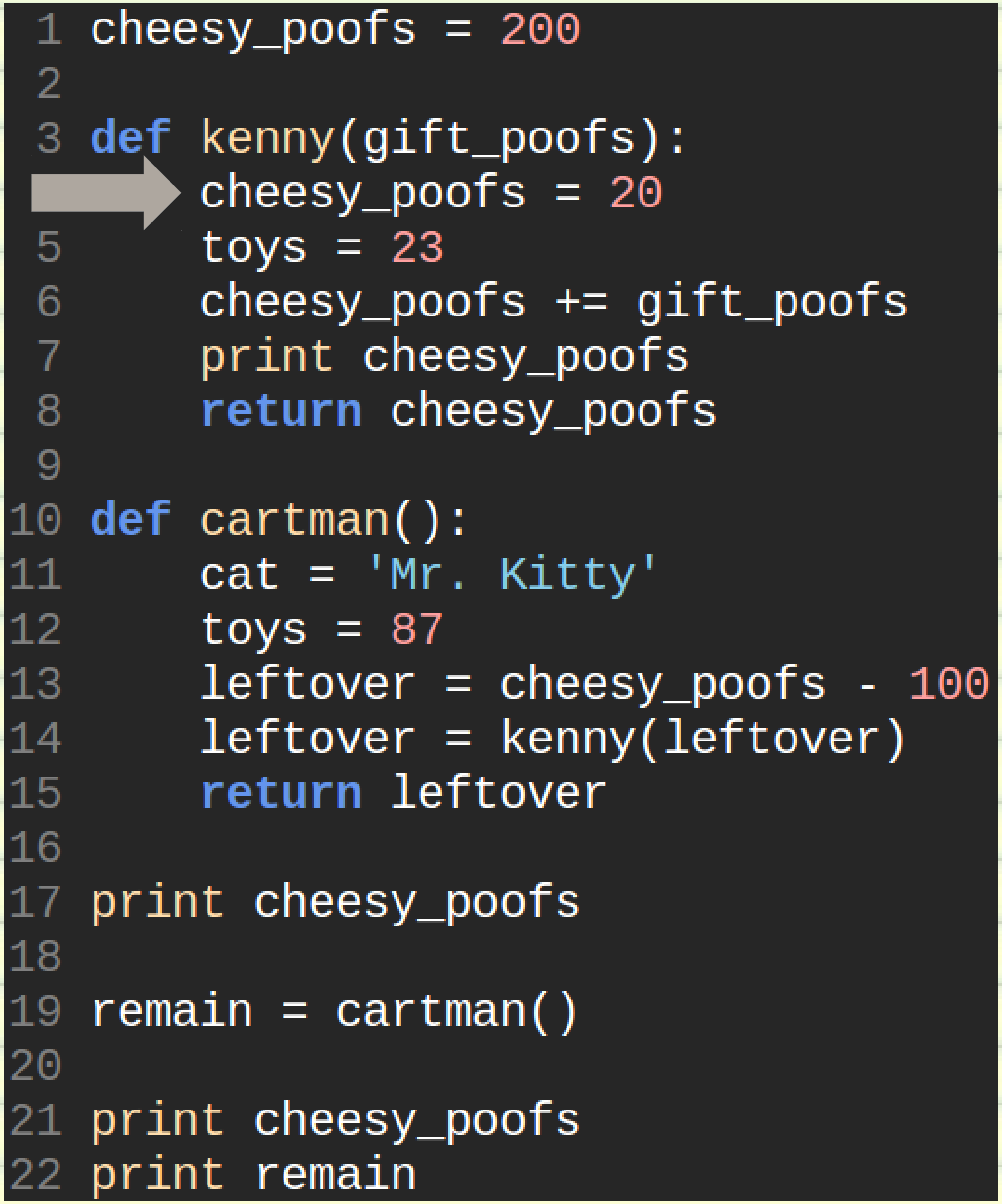
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

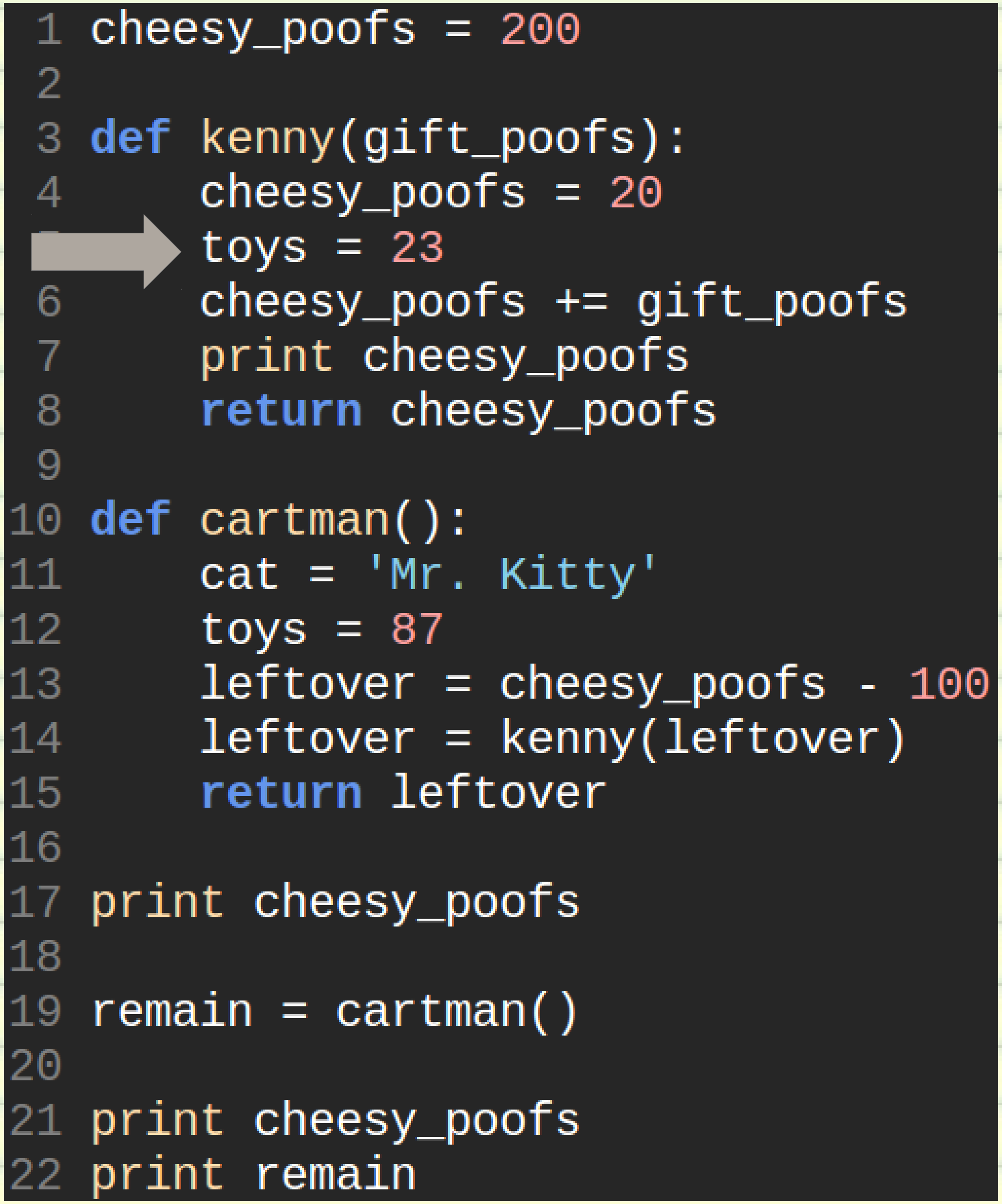
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

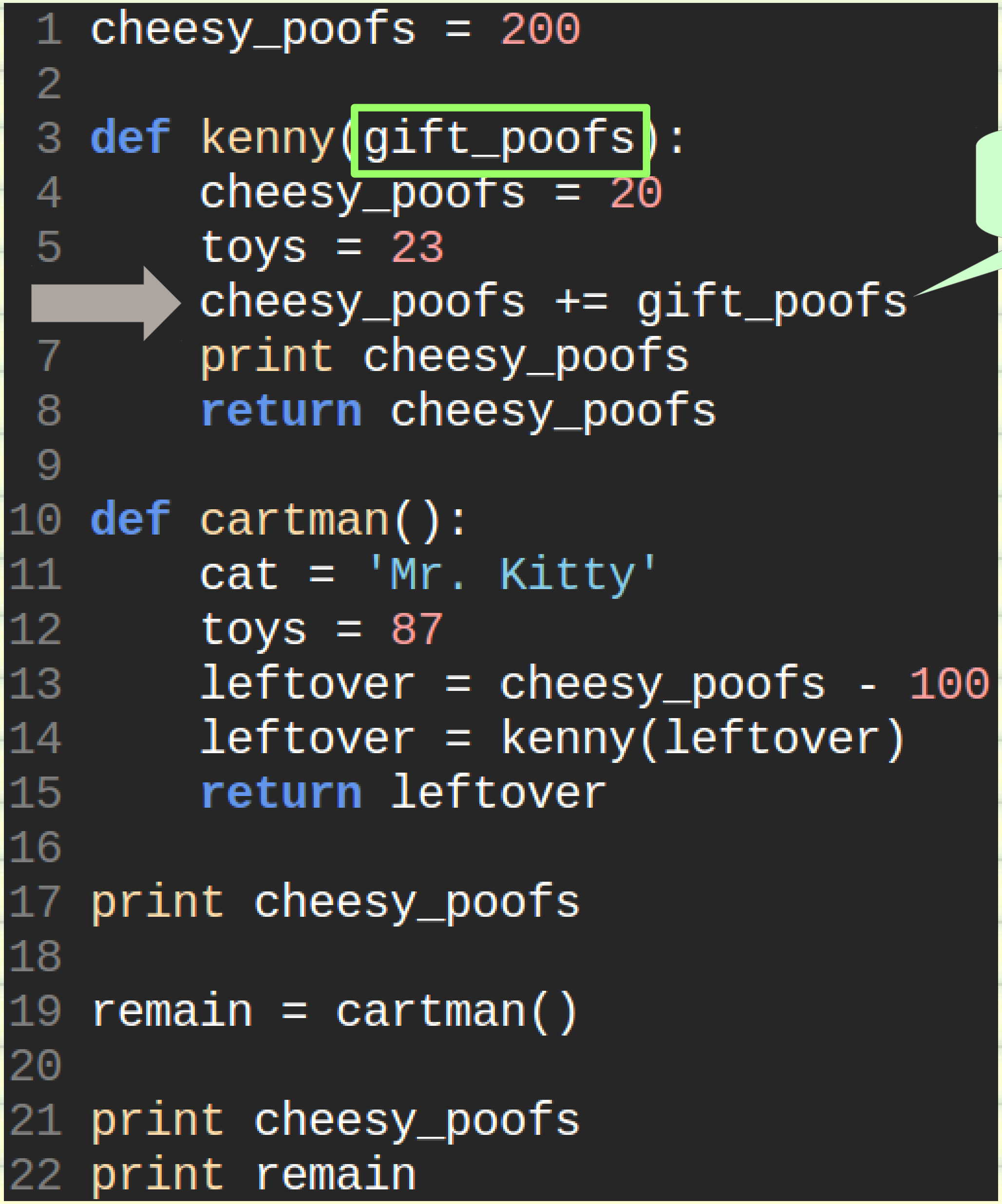
Namespaces & Variable Scope



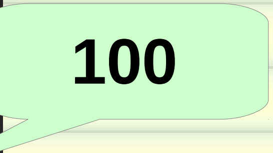
```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

Namespaces & Variable Scope

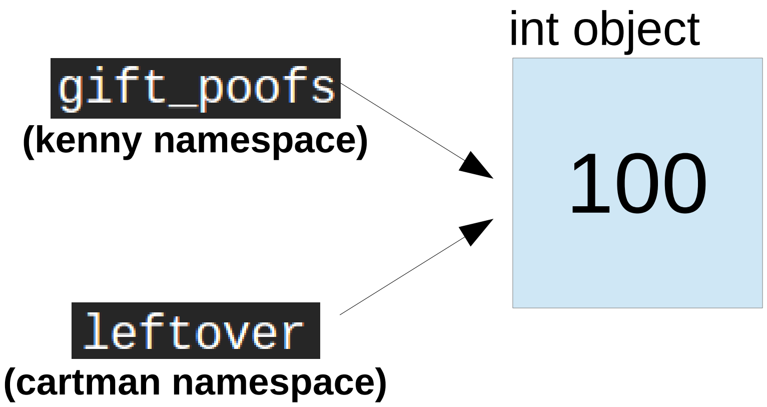


```python
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

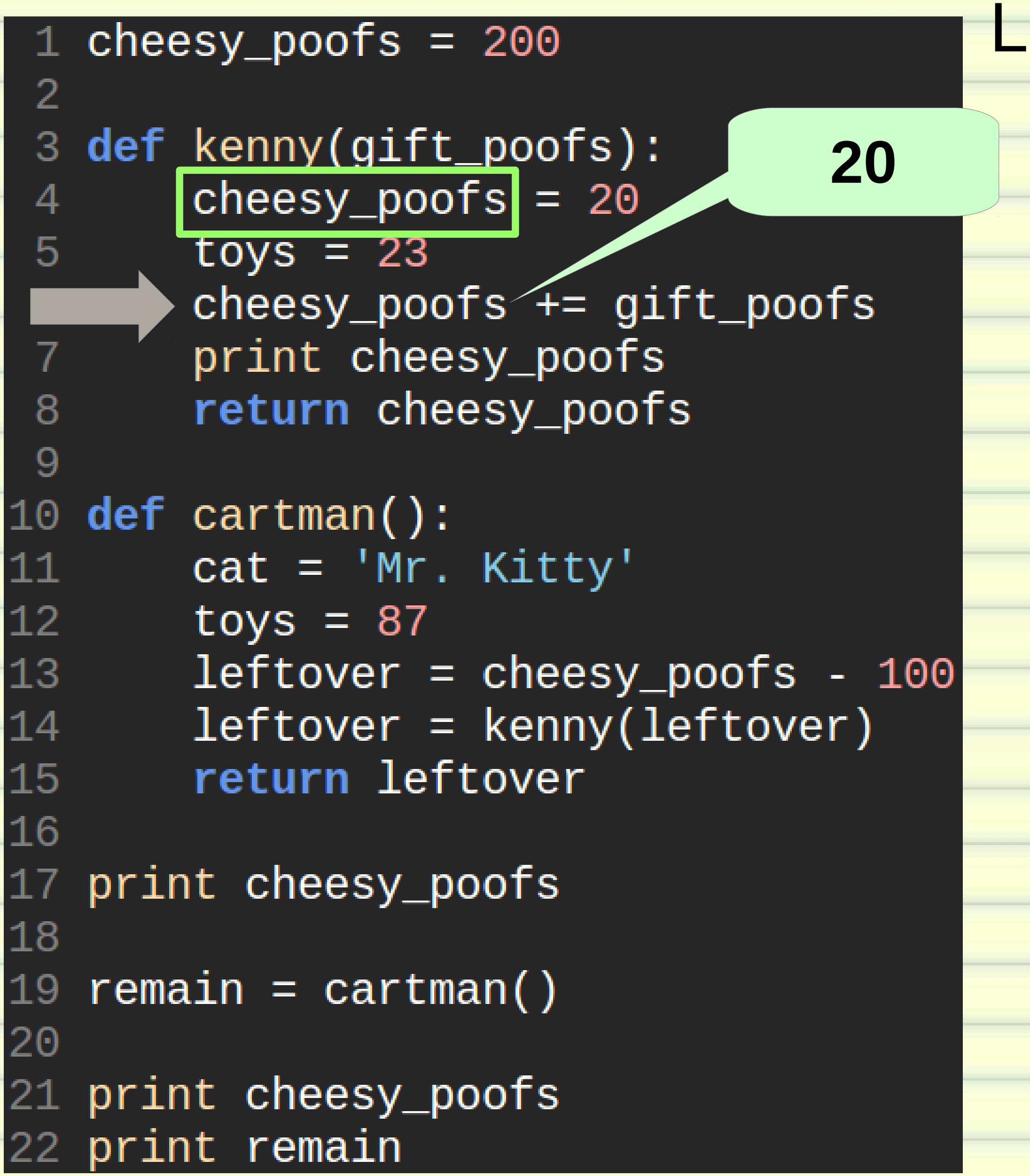


Let's trace this code.

Note:



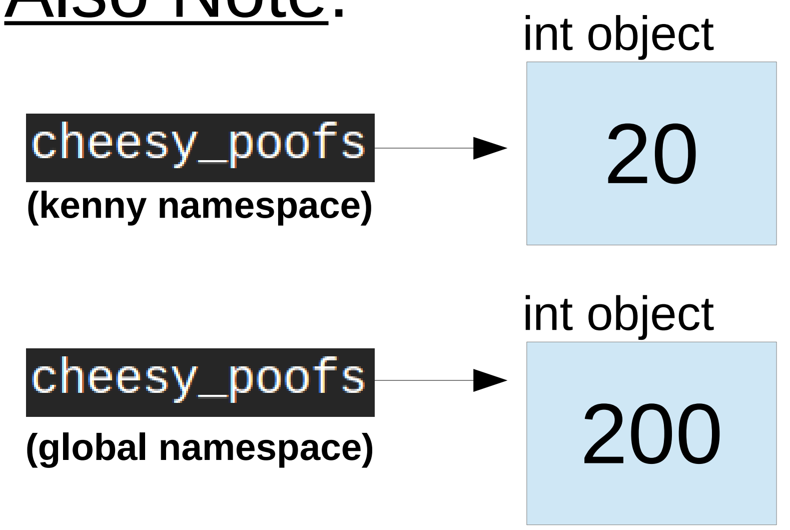
Namespaces & Variable Scope



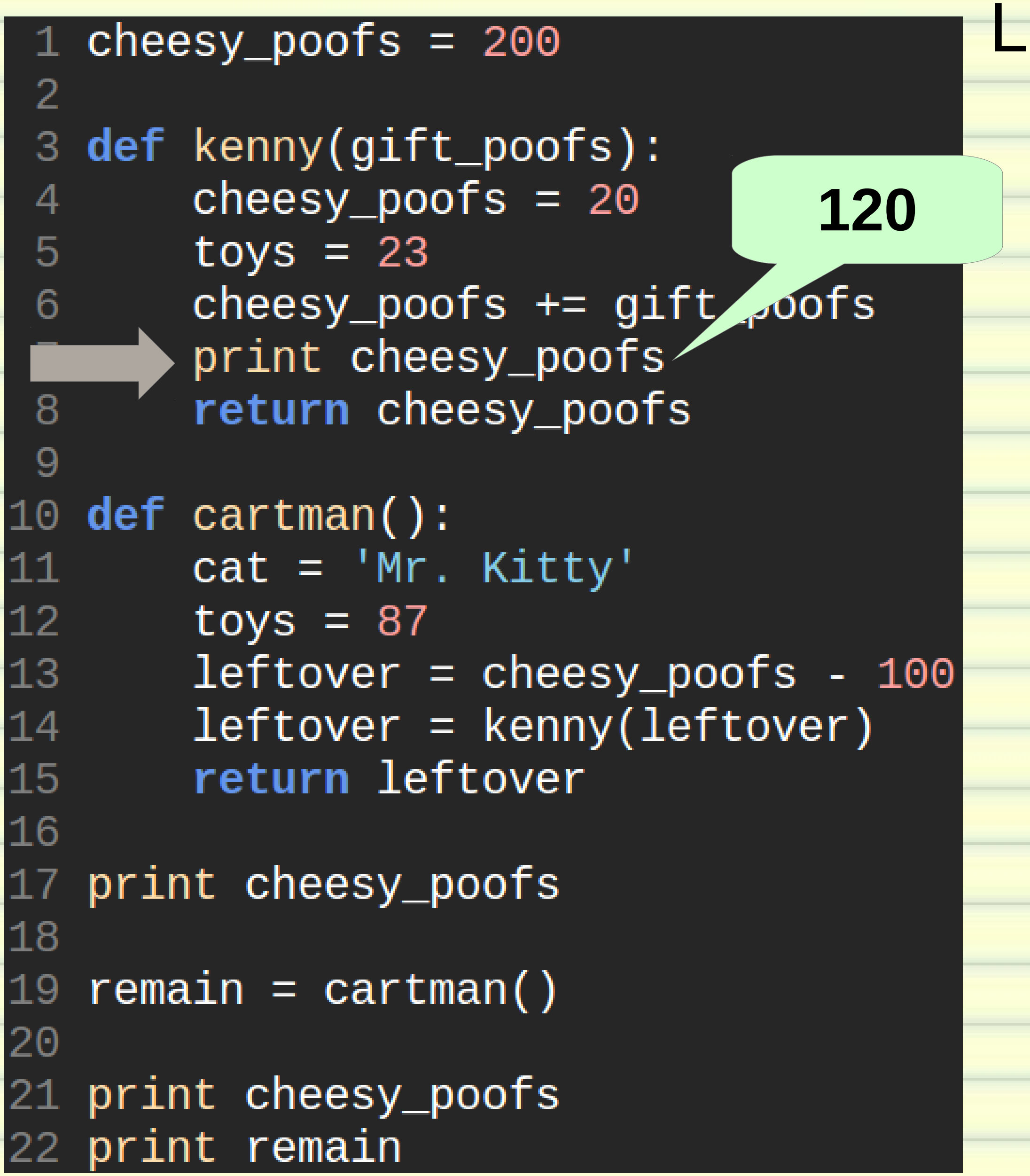
```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

Also Note:



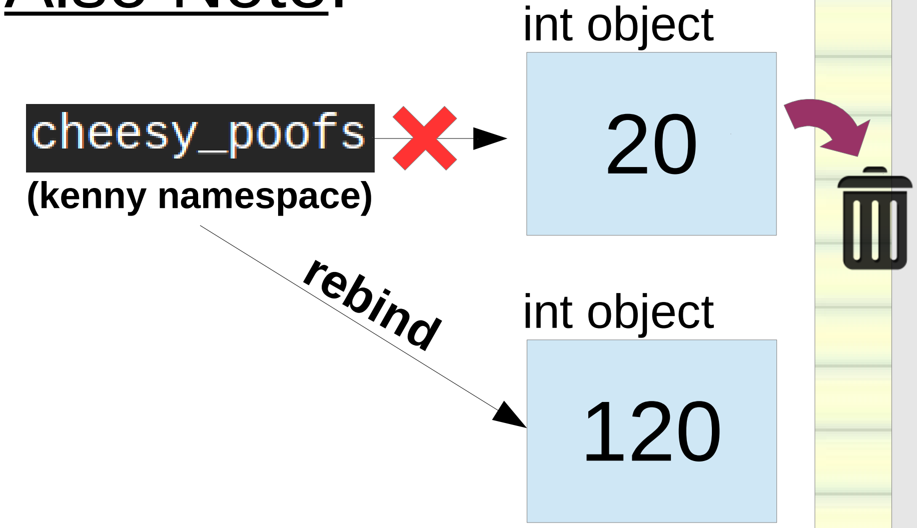
Namespaces & Variable Scope



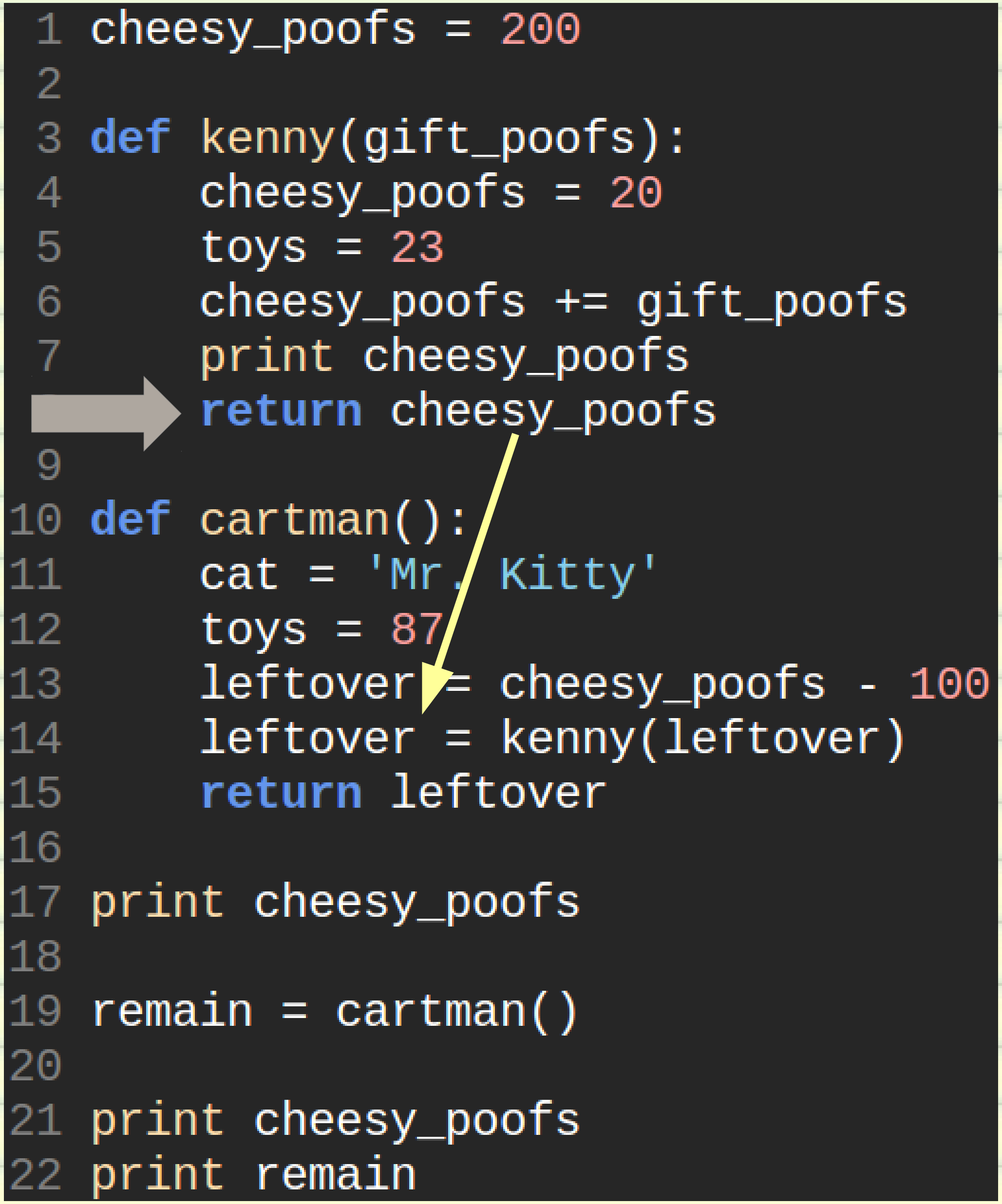
```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

Also Note:



Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

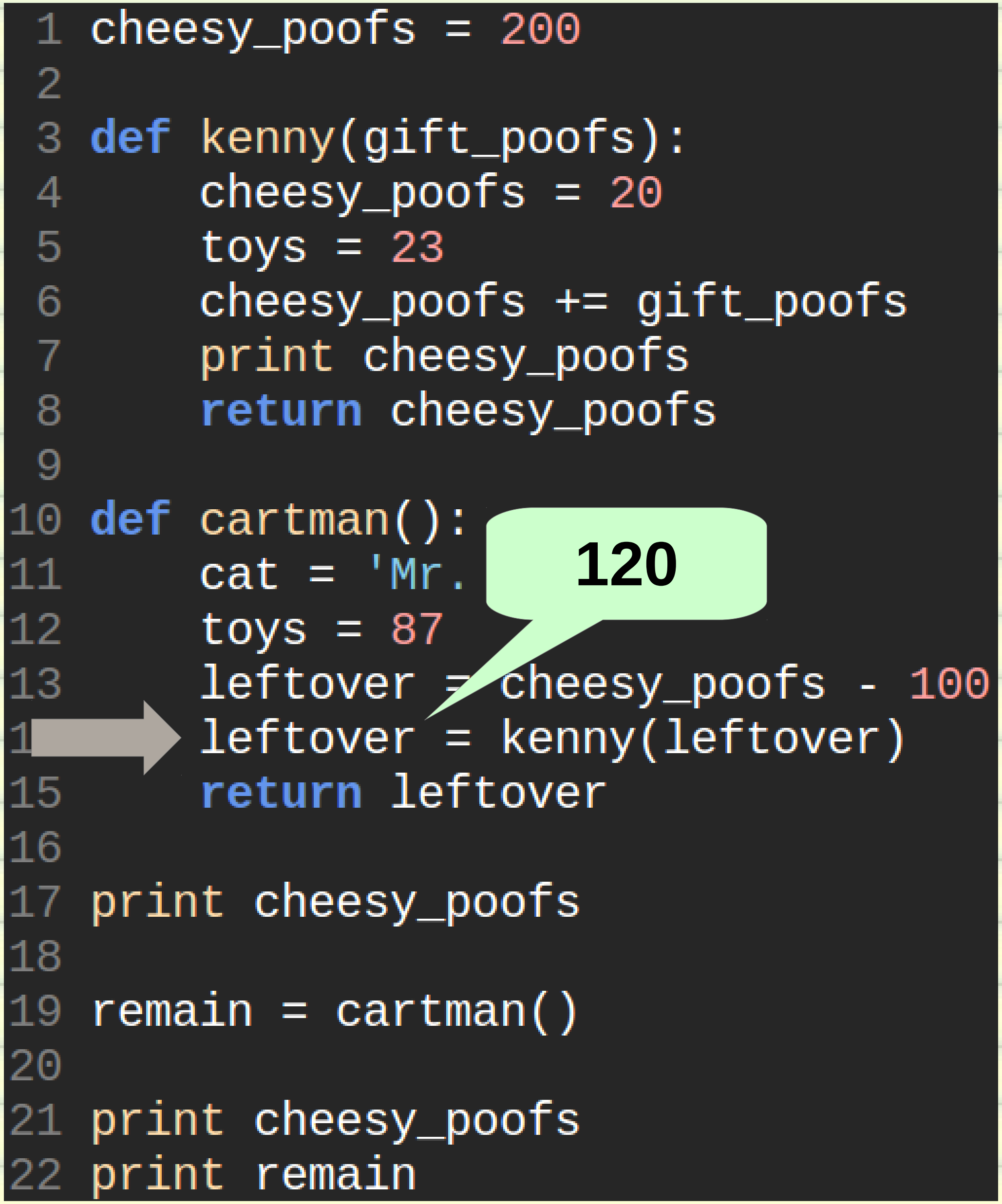
kenny's local namespace gets torn down.

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

the names are destroyed.

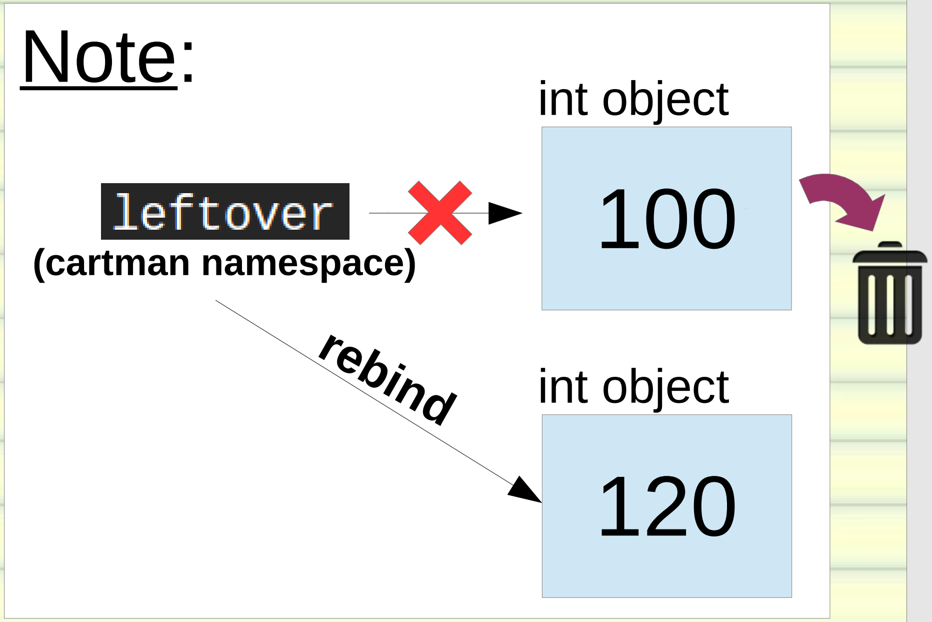
if this results in objects with 0 references, those objects are also destroyed

Namespaces & Variable Scope



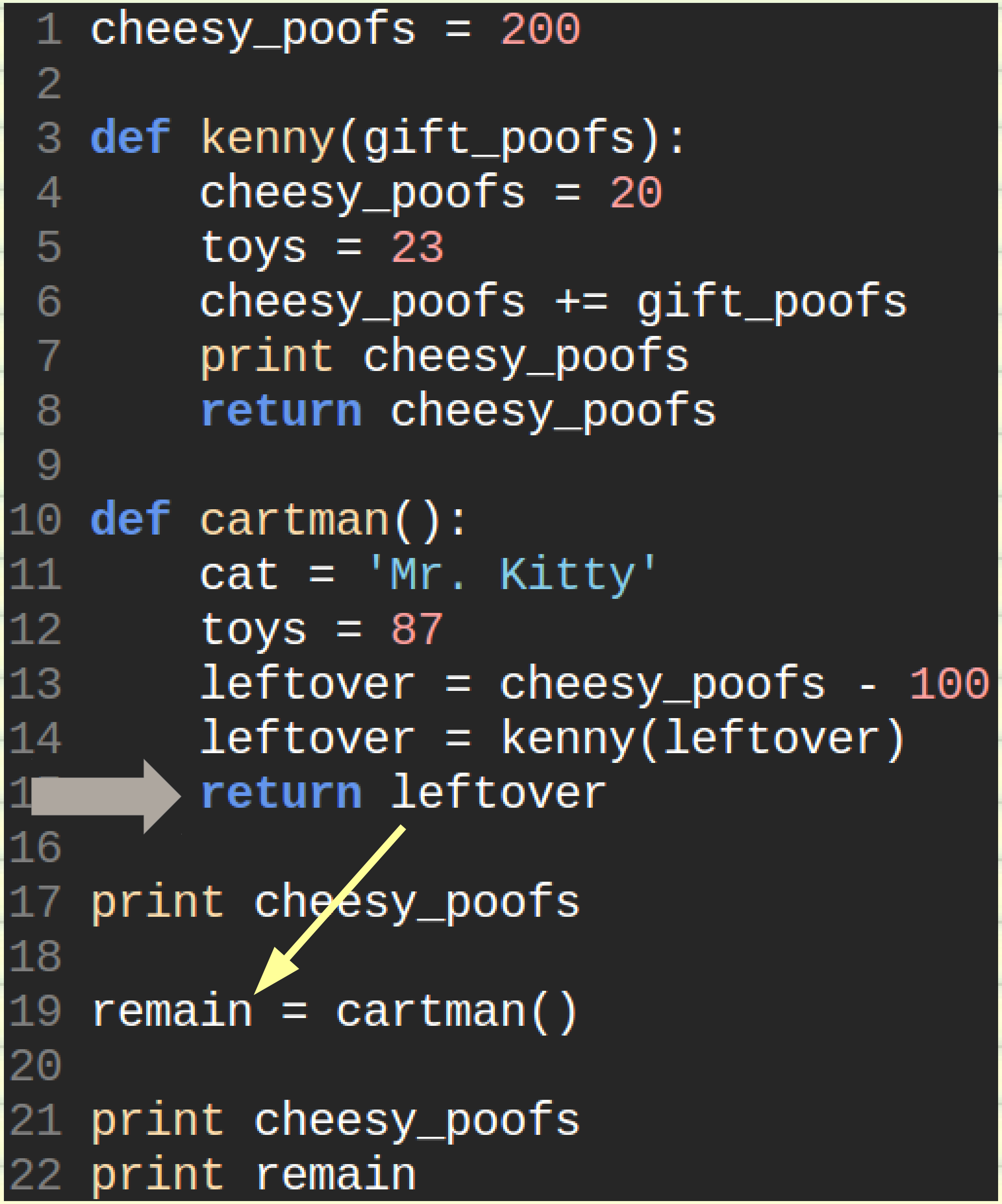
```
cheesy_poofs = 200

def kenny(gift_poofs):
    cheesy_poofs = 20
    toys = 23
    cheesy_poofs += gift_poofs
    print cheesy_poofs
    return cheesy_poofs

def cartman():
    cat = 'Mr.
    toys = 87
    leftover = cheesy_poofs - 100
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.



Note:

Namespaces & Variable Scope



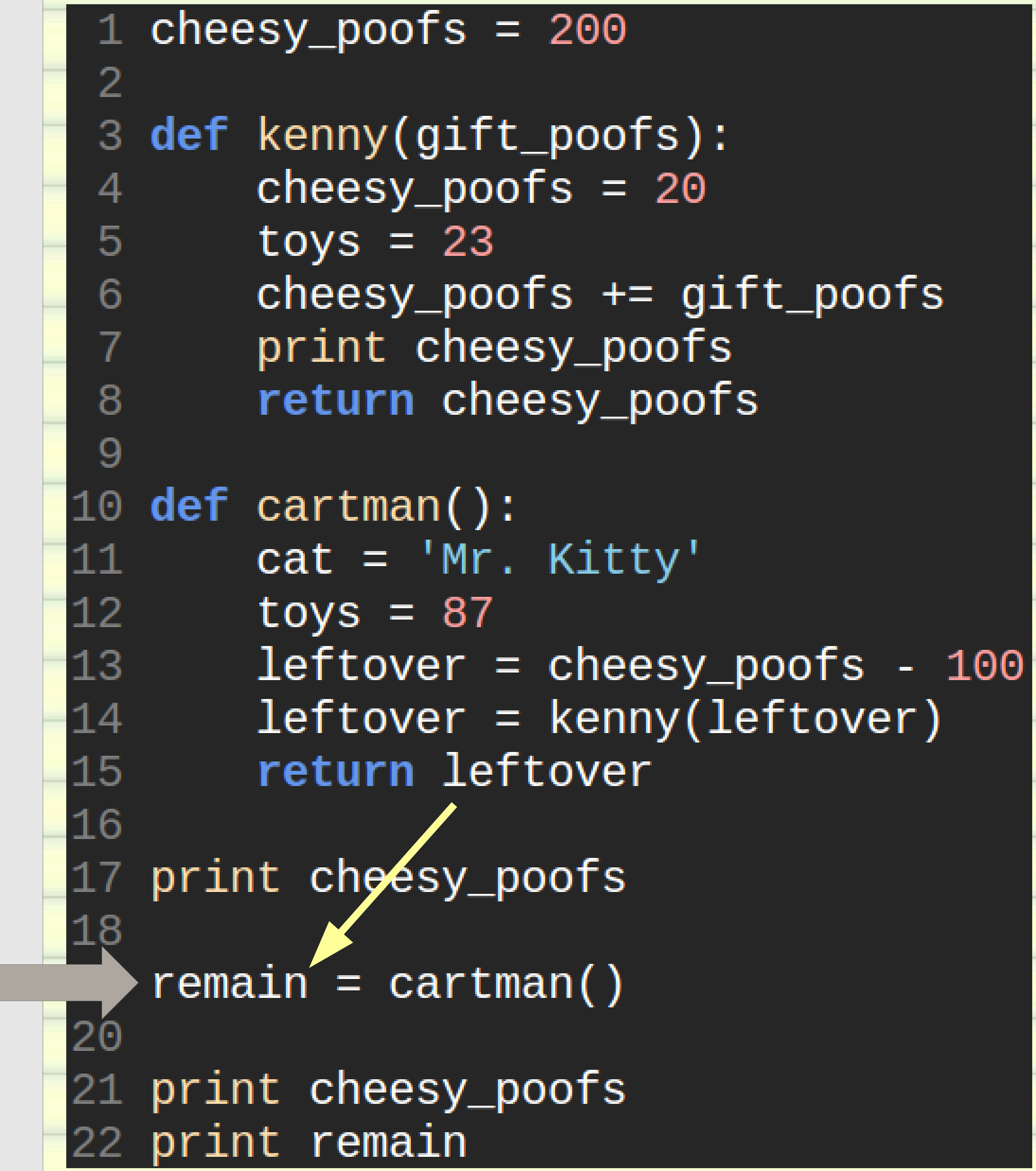
```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.

cartman's local namespace gets torn down.

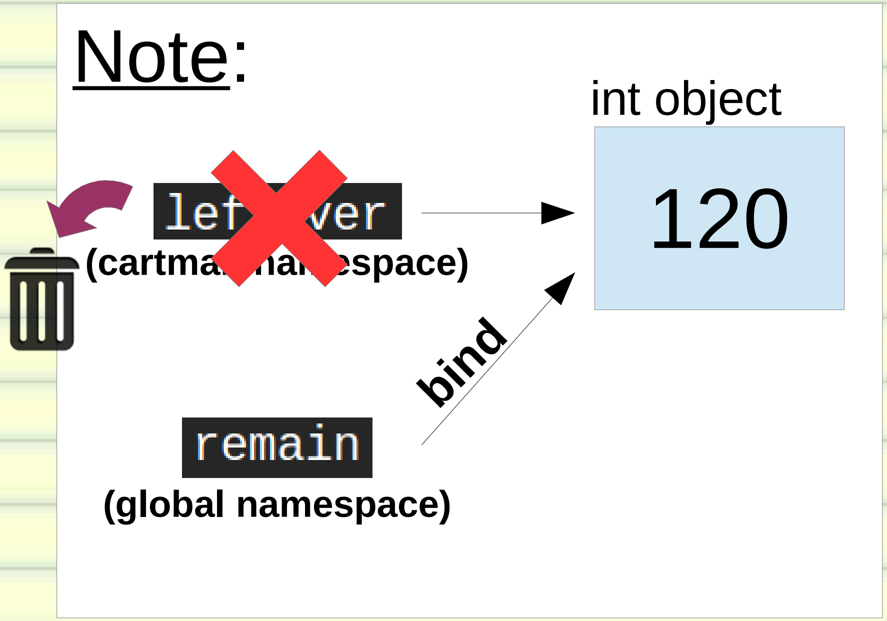
`['cat', 'leftover', 'toys']`

Namespaces & Variable Scope

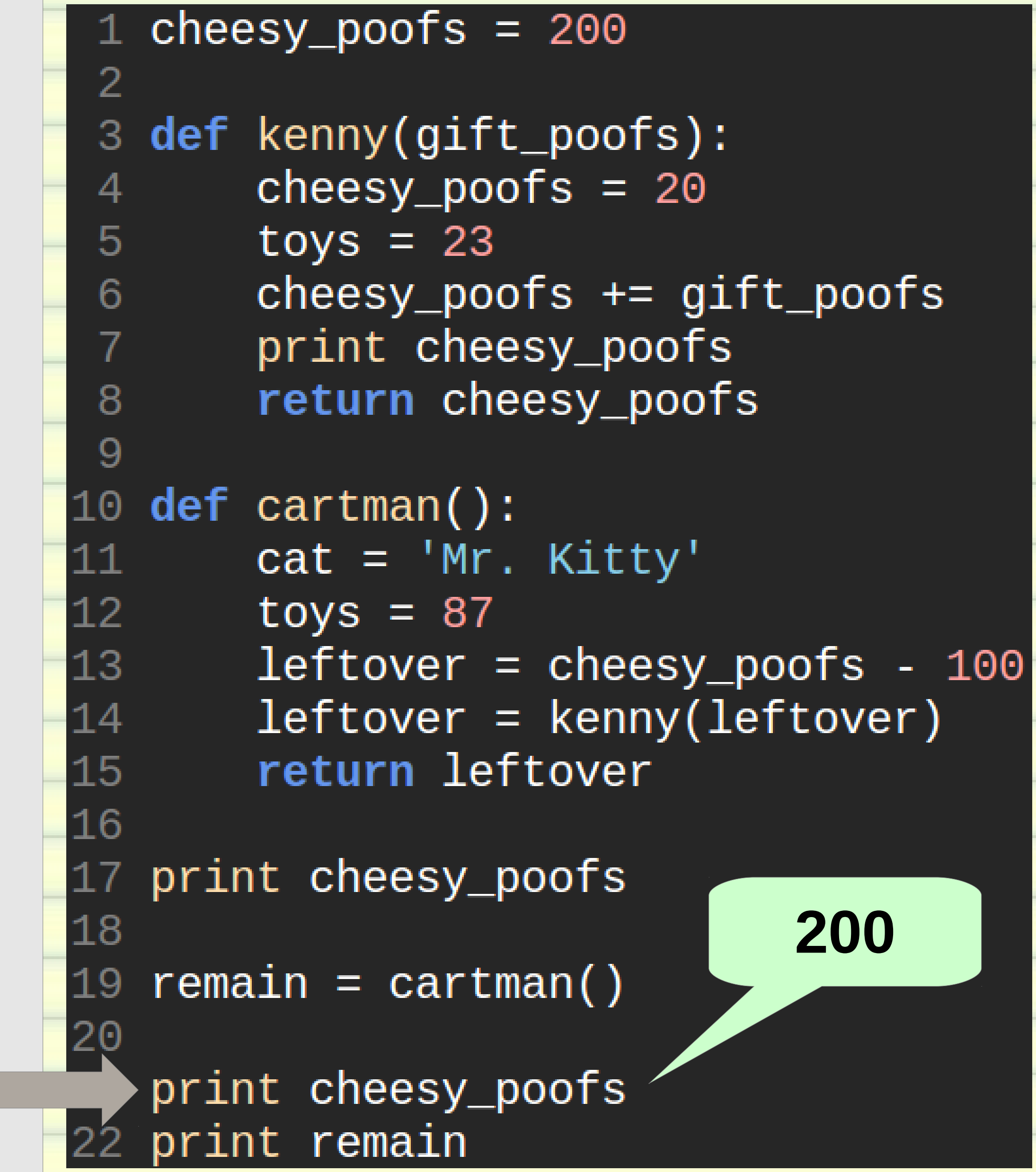


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

Let's trace this code.



Namespaces & Variable Scope

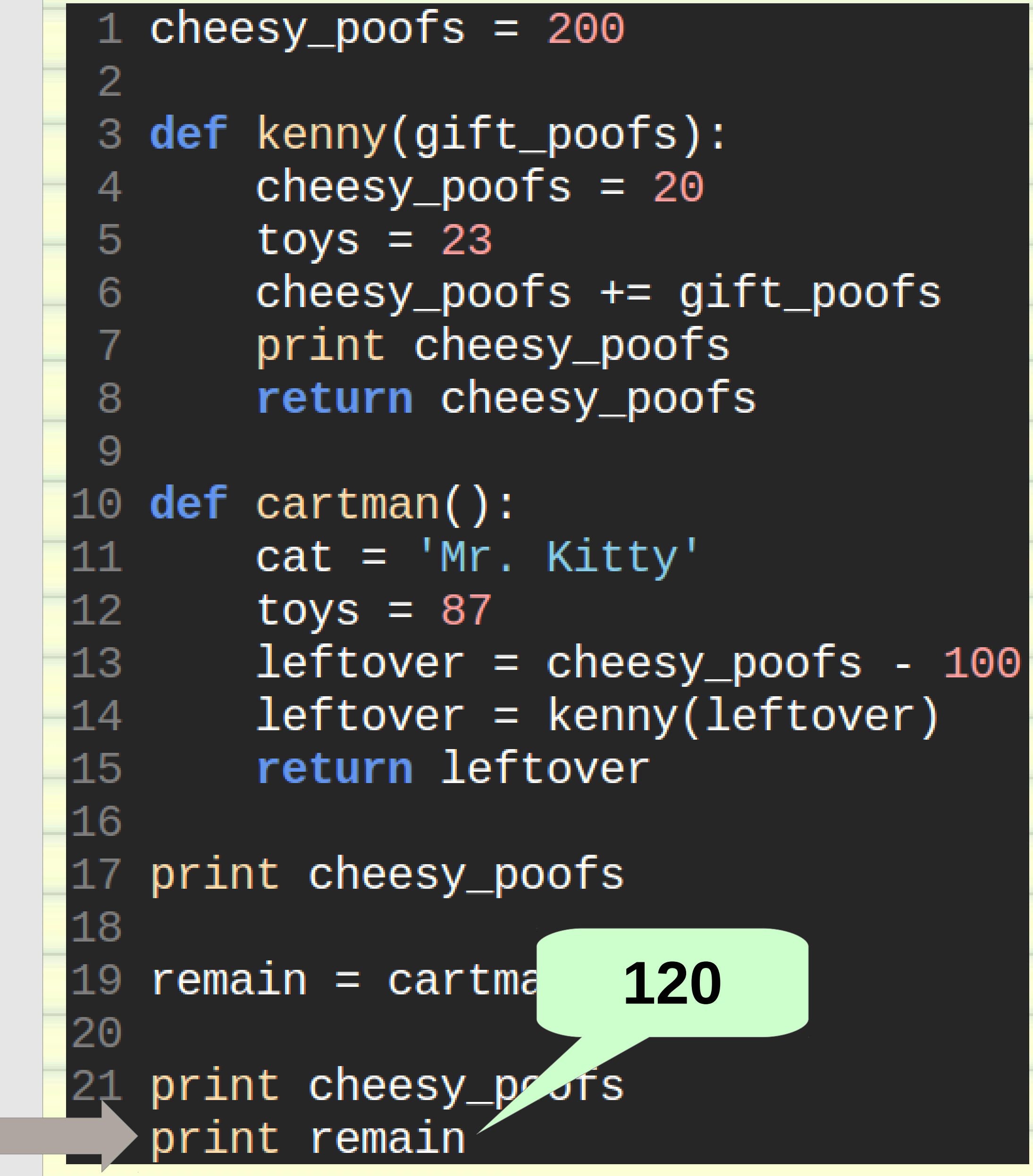


```
cheesy_poofs = 200

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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

200

Let's trace this code.

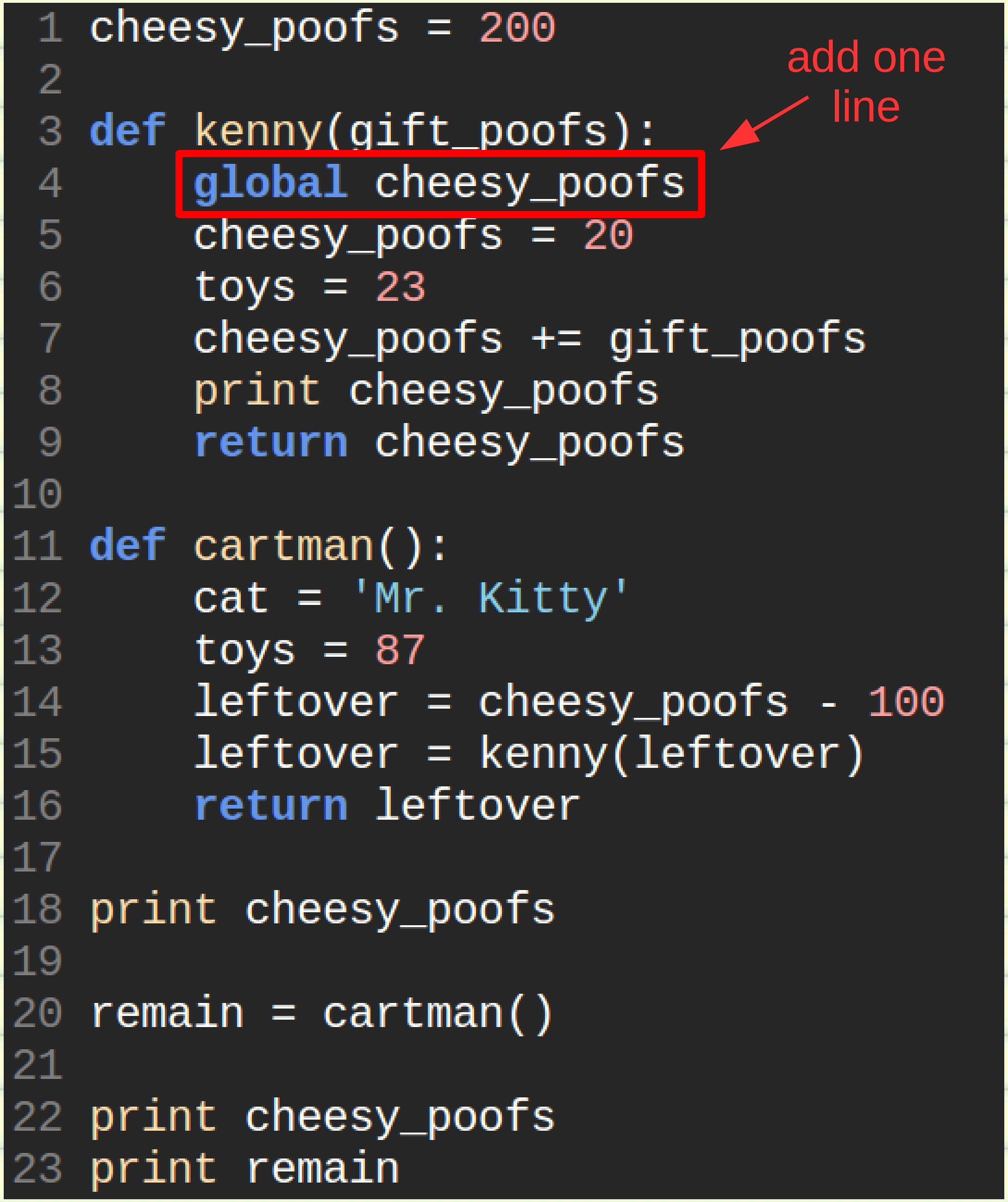
Namespaces & Variable Scope



```
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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartma

print cheesy_poofs
print remain
```

Let's trace this code.

Namespaces & Variable Scope



```
cheesy_poofs = 200

def kenny(gift_poofs):
    global cheesy_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
    leftover = kenny(leftover)
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

add one line

The **global** keyword...

what does it do?

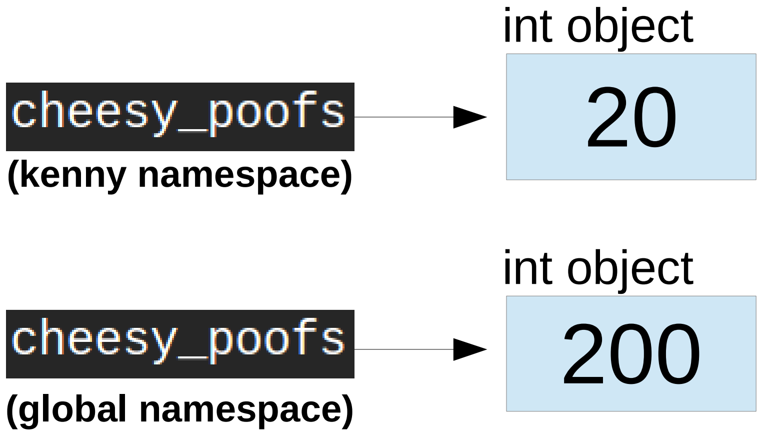
Namespaces & Variable Scope

```
 1 cheesy_poofs = 200
 2
 3 def kenny(gift_poofs):
 4     global cheesy_poofs
       cheesy_poofs = 20
 6     toys = 23
 7     cheesy_poofs += gift_poofs
 8     print cheesy_poofs
 9     return cheesy_poofs
10
11 def cartman():
12     cat = 'Mr. Kitty'
13     toys = 87
14     leftover = cheesy_poofs - 100
15     leftover = kenny(leftover)
16     return leftover
17
18 print cheesy_poofs
19
20 remain = cartman()
21
22 print cheesy_poofs
23 print remain
```

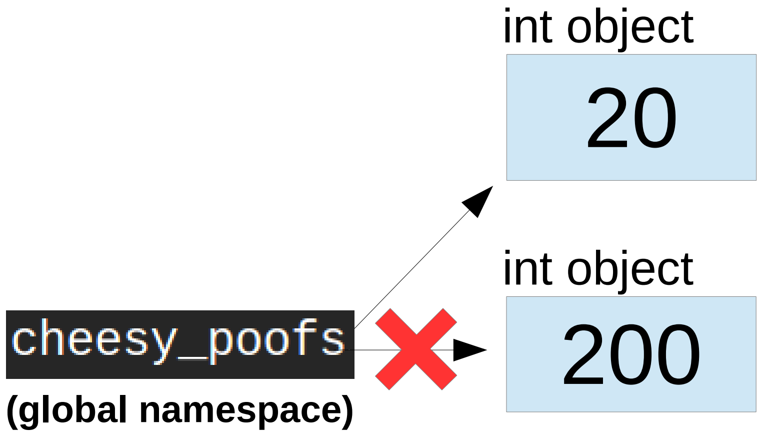
add one line

The **global** keyword...

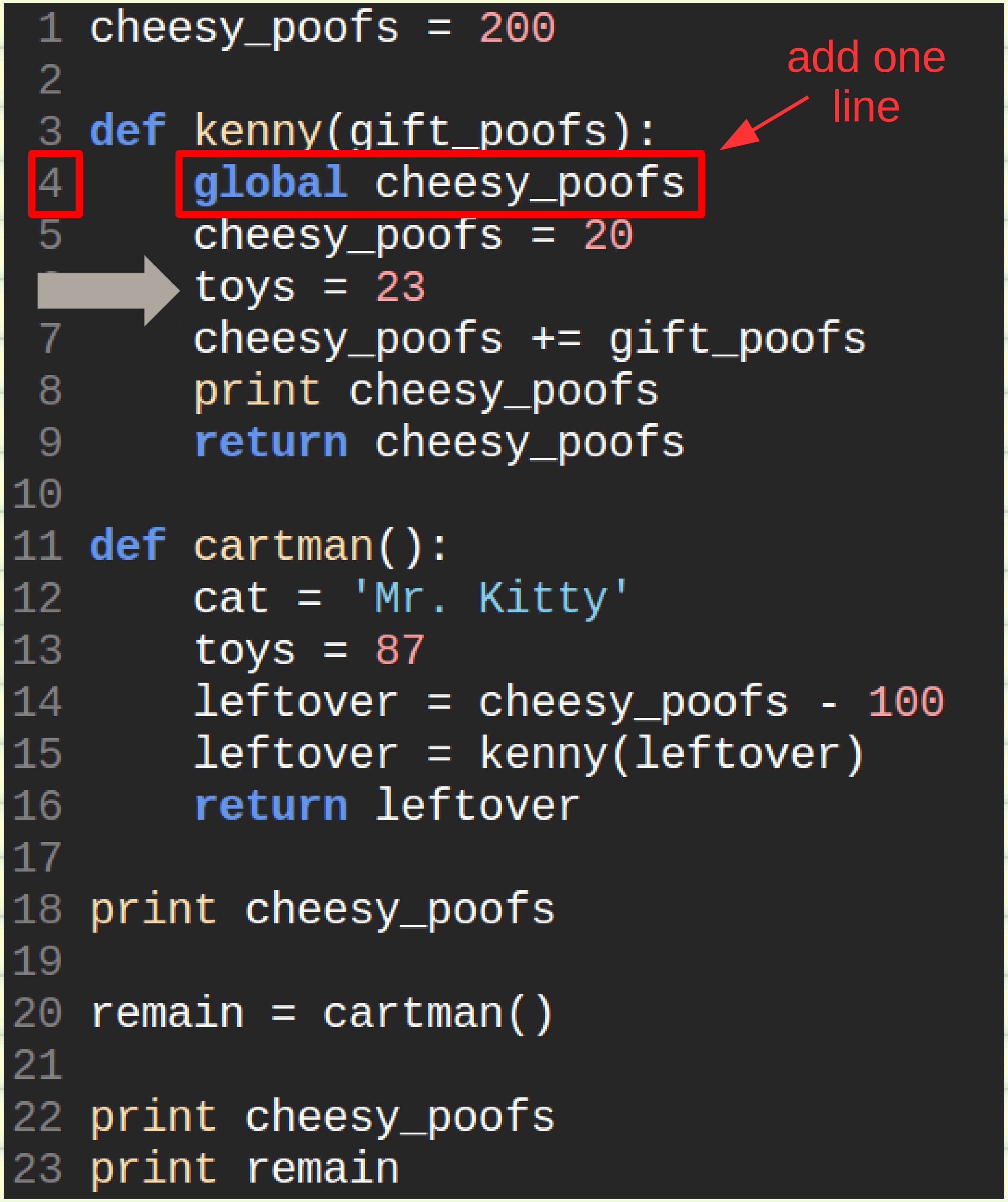
Without **line 4**



***With* line 4**



Namespaces & Variable Scope

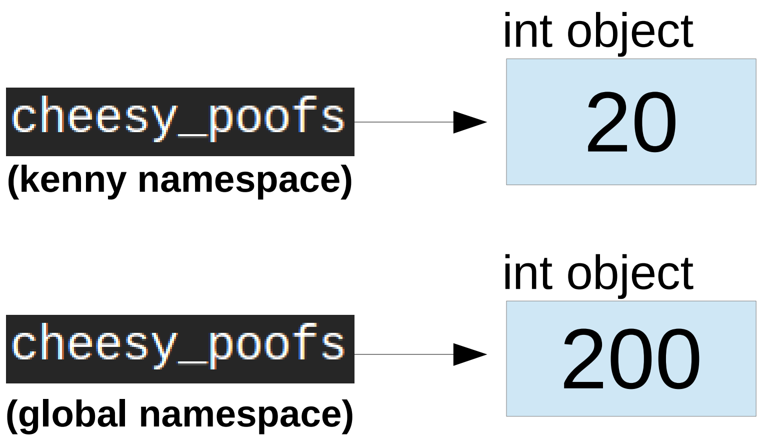


```
 1 cheesy_poofs = 200
 2
 3 def kenny(gift_poofs):
 4     global cheesy_poofs
 5     cheesy_poofs = 20
 6     toys = 23
 7     cheesy_poofs += gift_poofs
 8     print cheesy_poofs
 9     return cheesy_poofs
10
11 def cartman():
12     cat = 'Mr. Kitty'
13     toys = 87
14     leftover = cheesy_poofs - 100
15     leftover = kenny(leftover)
16     return leftover
17
18 print cheesy_poofs
19
20 remain = cartman()
21
22 print cheesy_poofs
23 print remain
```

add one line

The **global** keyword...

Without **line 4**



int object

cheesy_poofs

(kenny namespace)

20

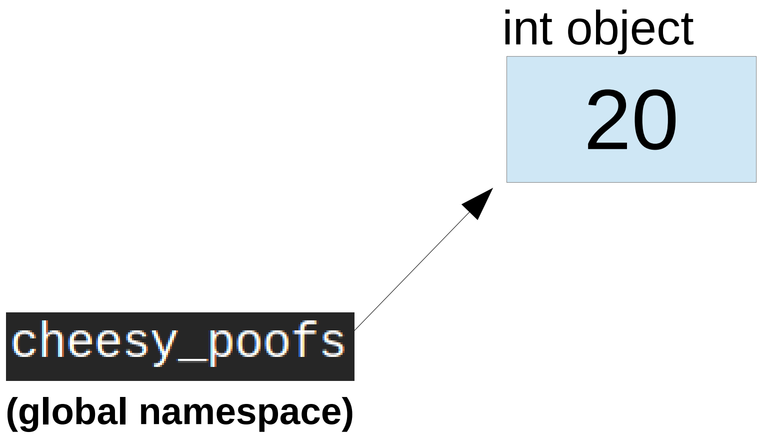
int object

cheesy_poofs

(global namespace)

200

With **line 4**



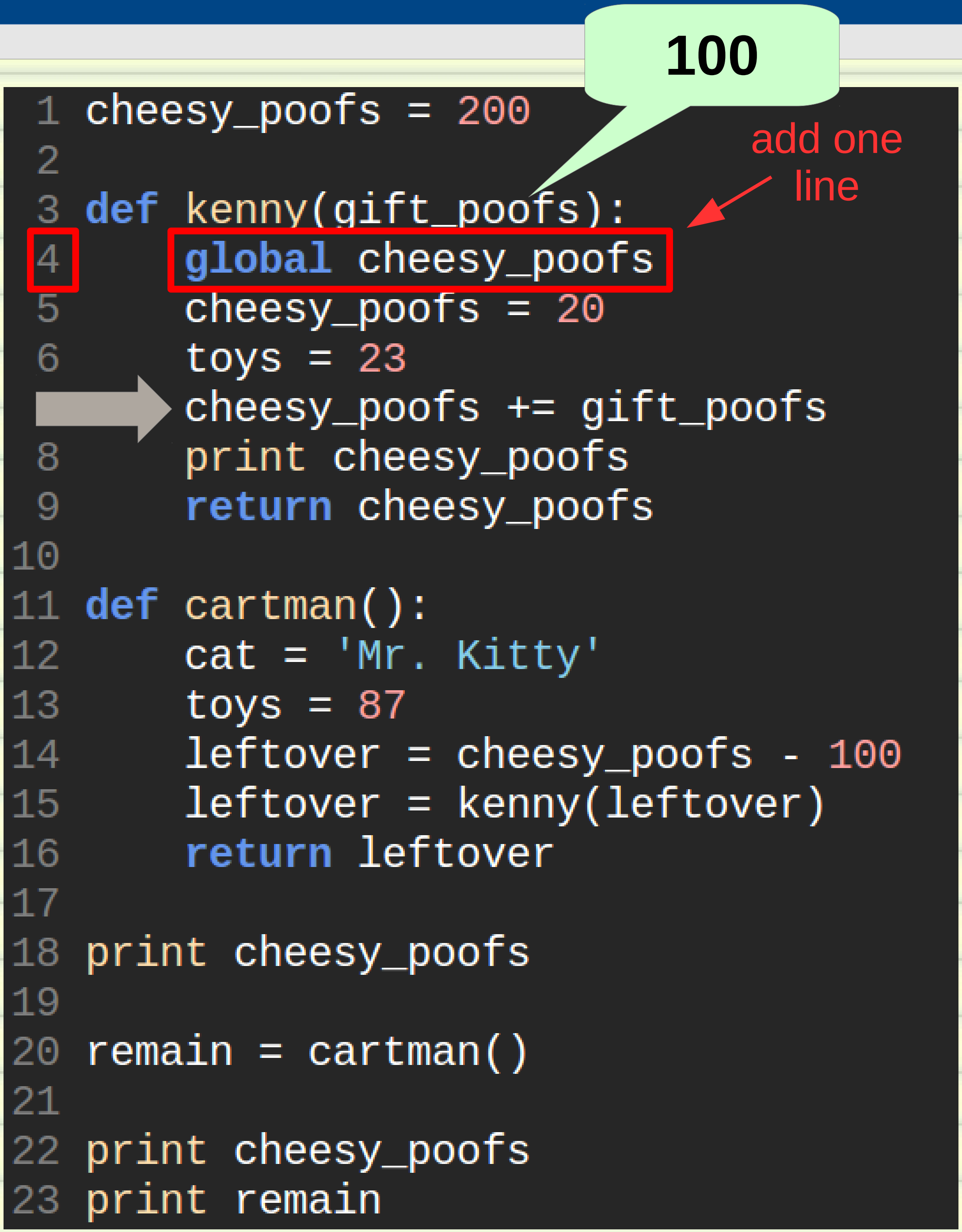
int object

20

cheesy_poofs

(global namespace)

Namespaces & Variable Scope

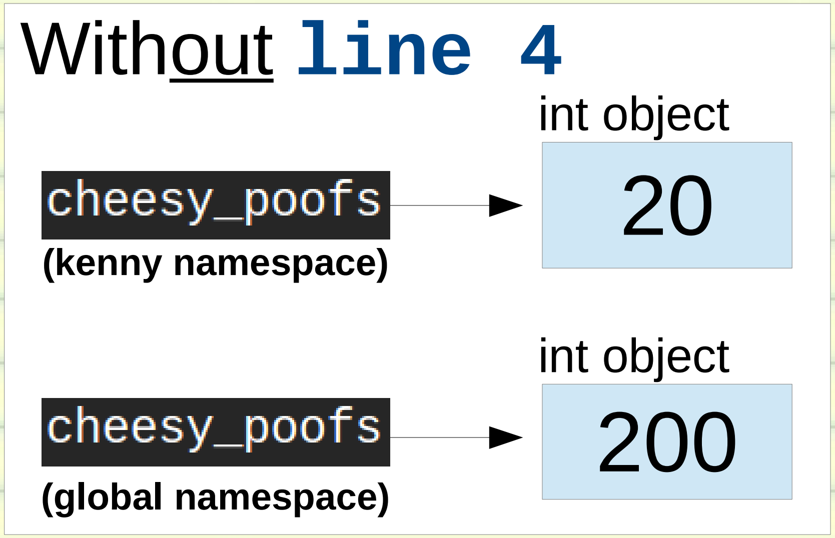


```
 1 cheesy_poofs = 200
 2
 3 def kenny(gift_poofs):
 4     global cheesy_poofs
 5     cheesy_poofs = 20
 6     toys = 23
       cheesy_poofs += gift_poofs
 8     print cheesy_poofs
 9     return cheesy_poofs
10
11 def cartman():
12     cat = 'Mr. Kitty'
13     toys = 87
14     leftover = cheesy_poofs - 100
15     leftover = kenny(leftover)
16     return leftover
17
18 print cheesy_poofs
19
20 remain = cartman()
21
22 print cheesy_poofs
23 print remain
```

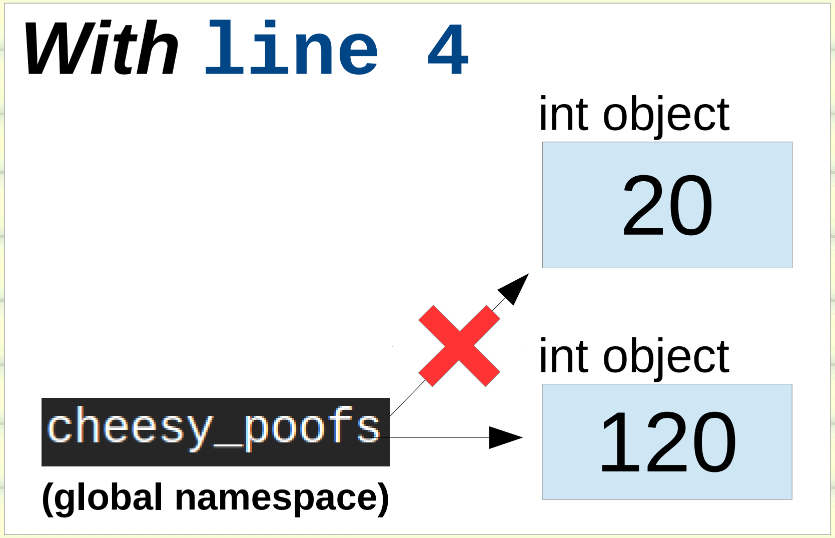
100

add one line

The **global** keyword...



Without **line 4**



***With* line 4**

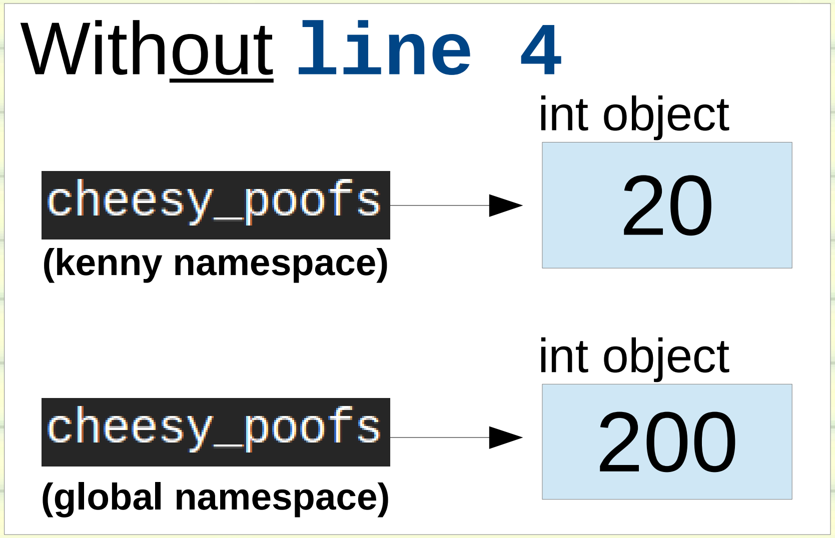
Namespaces & Variable Scope

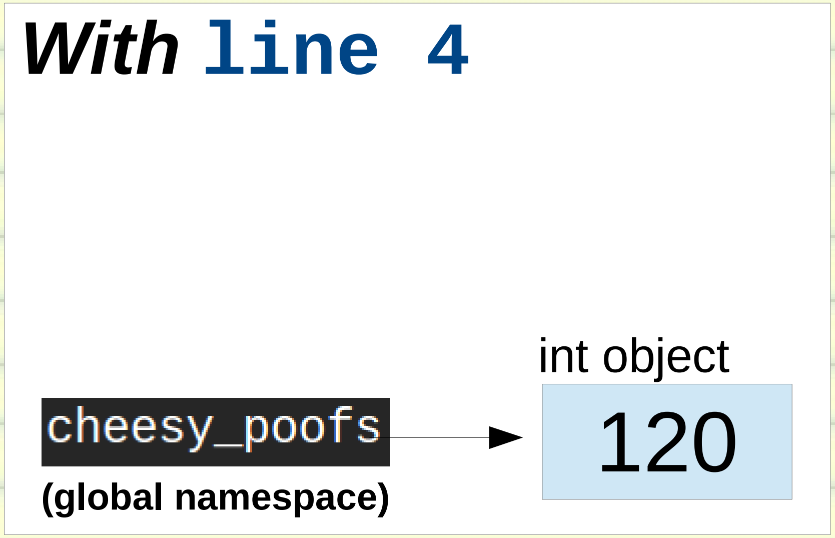
```
 1 cheesy_poofs = 200
 2
 3 def kenny(gift_poofs):
 4     global cheesy_poofs
 5     cheesy_poofs = 20
 6     toys = 23
 7     cheesy_poofs += gift_poofs
       print cheesy_poofs
 9     return cheesy_poofs
10
11 def cartman():
12     cat = 'Mr. Kitty'
13     toys = 87
14     leftover = cheesy_poofs - 100
15     leftover = kenny(leftover)
16     return leftover
17
18 print cheesy_poofs
19
20 remain = cartman()
21
22 print cheesy_poofs
23 print remain
```

add one line

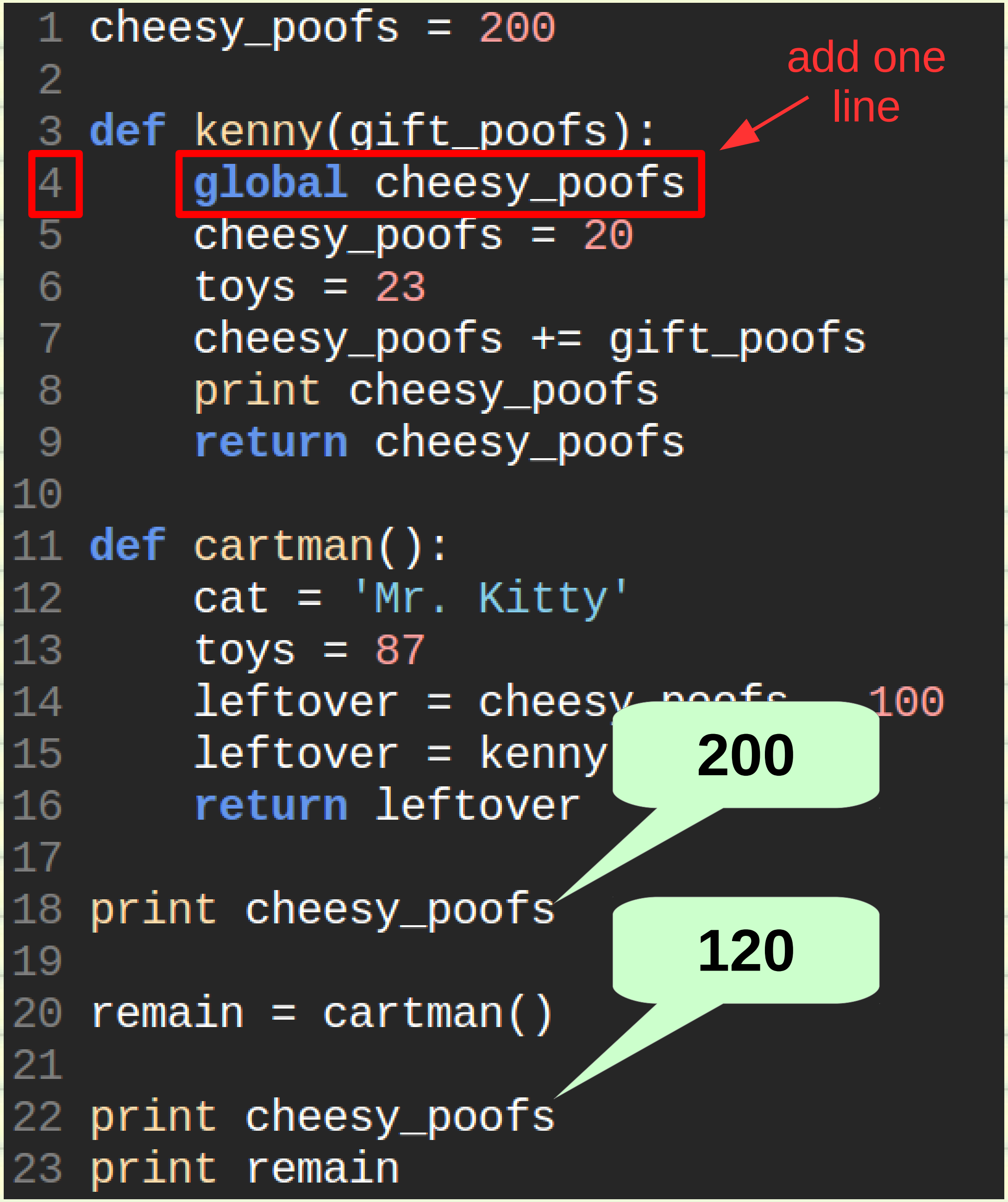
120

The **global** keyword...





Namespaces & Variable Scope



```
cheesy_poofs = 200

def kenny(gift_poofs):
    global cheesy_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
    leftover = kenny
    return leftover

print cheesy_poofs

remain = cartman()

print cheesy_poofs
print remain
```

add one line

200

120

The **global** keyword...

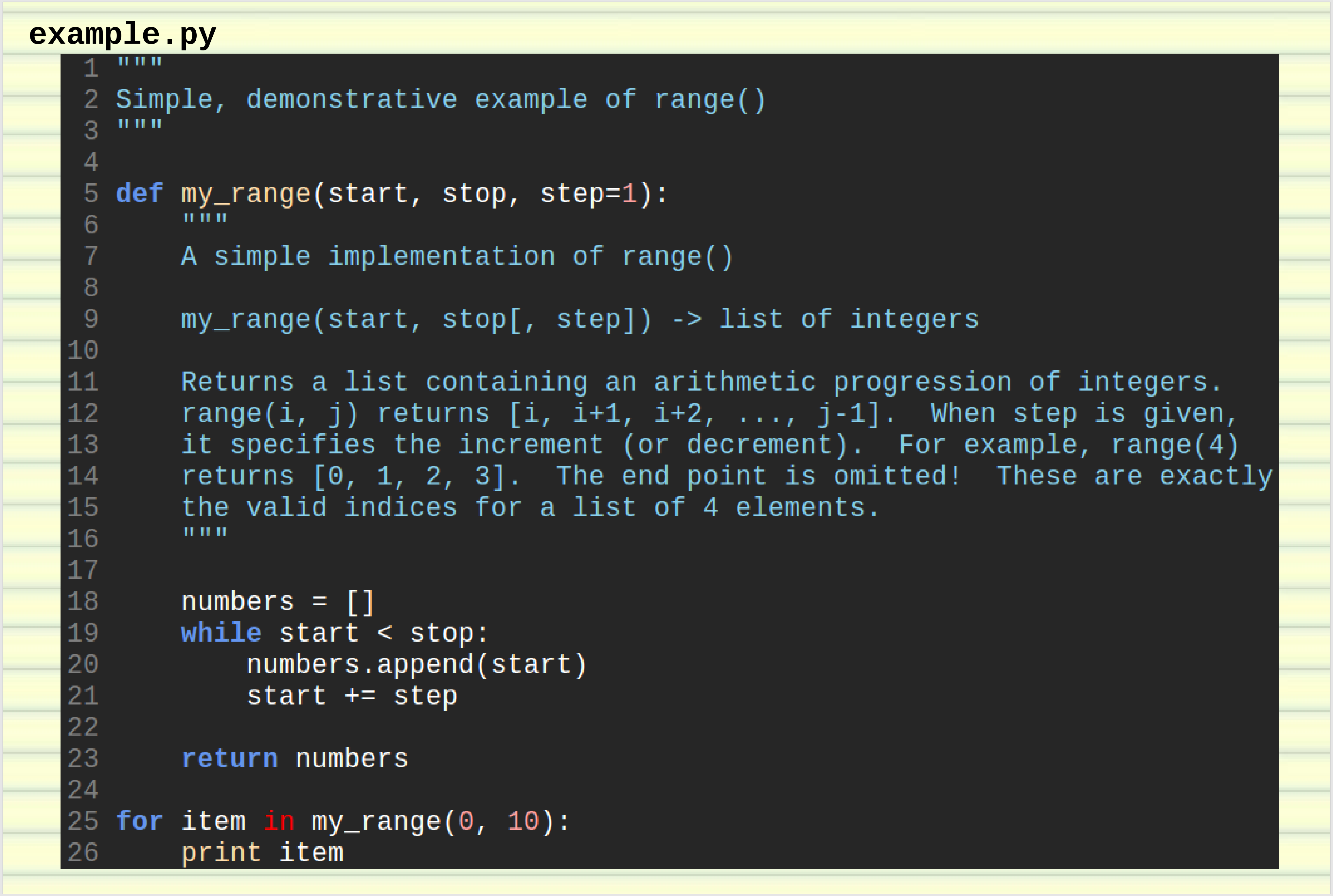
lets you manipulate a global name inside your function's namespace

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

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

`range()`

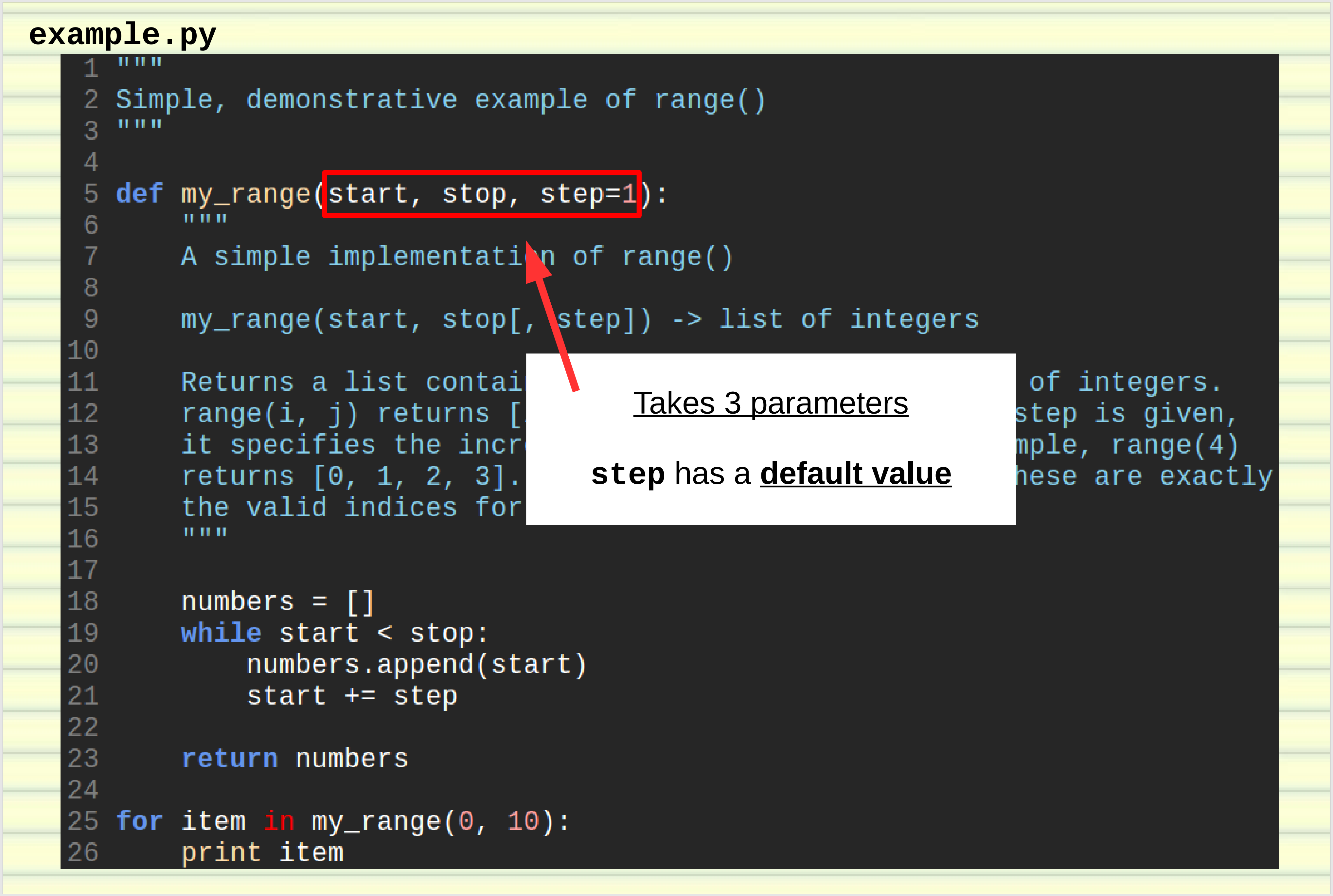
Using Functions



example.py

```python
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

Using Functions



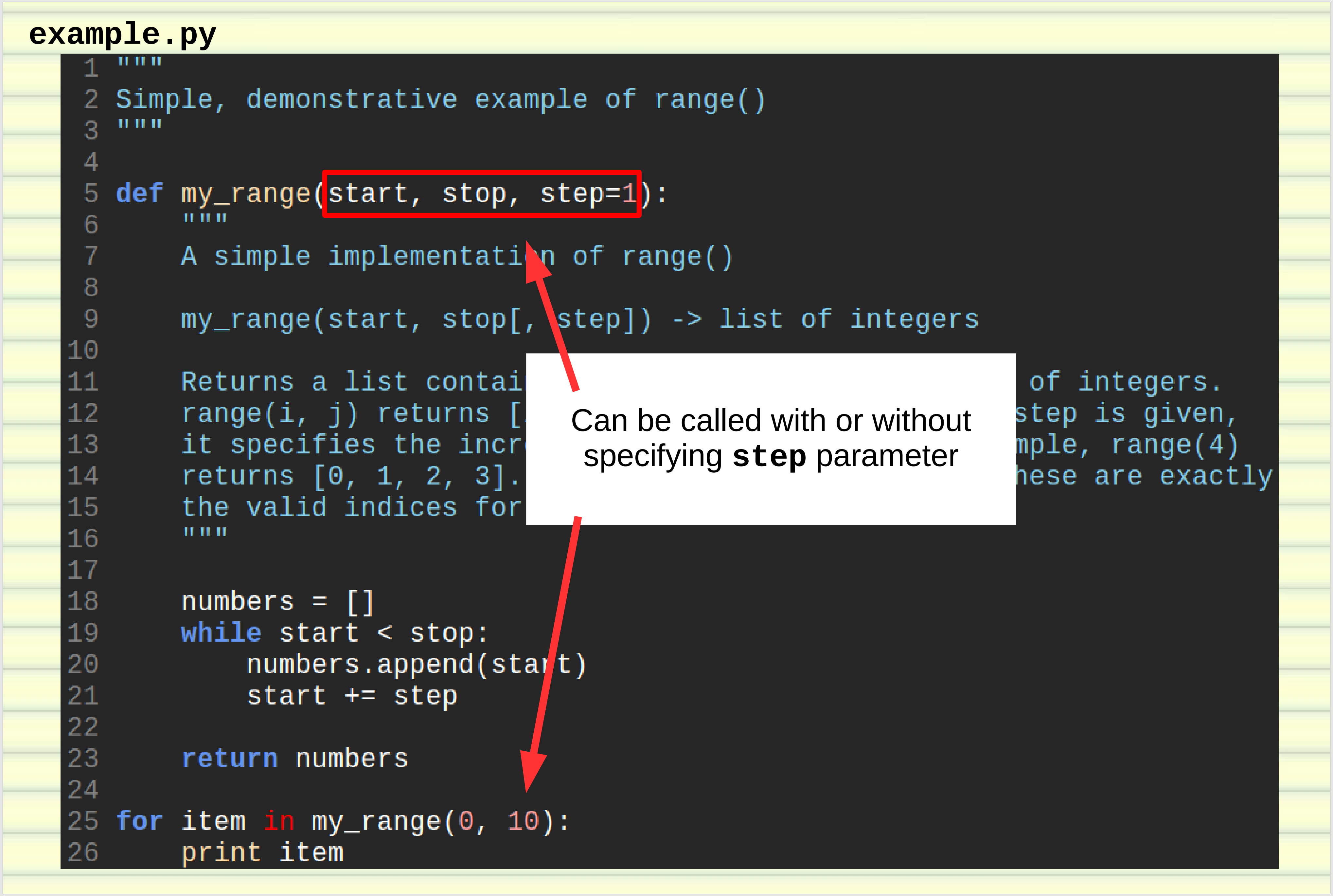
example.py

```
"""
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

    Returns a list contai                               of integers.
    range(i, j) returns [                              step is given,
    it specifies the incr                              mple, range(4)
    returns [0, 1, 2, 3].                              hese are exactly
    the valid indices for
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

<u>Takes 3 parameters</u>

`step` has a **<u>default value</u>**

Using Functions

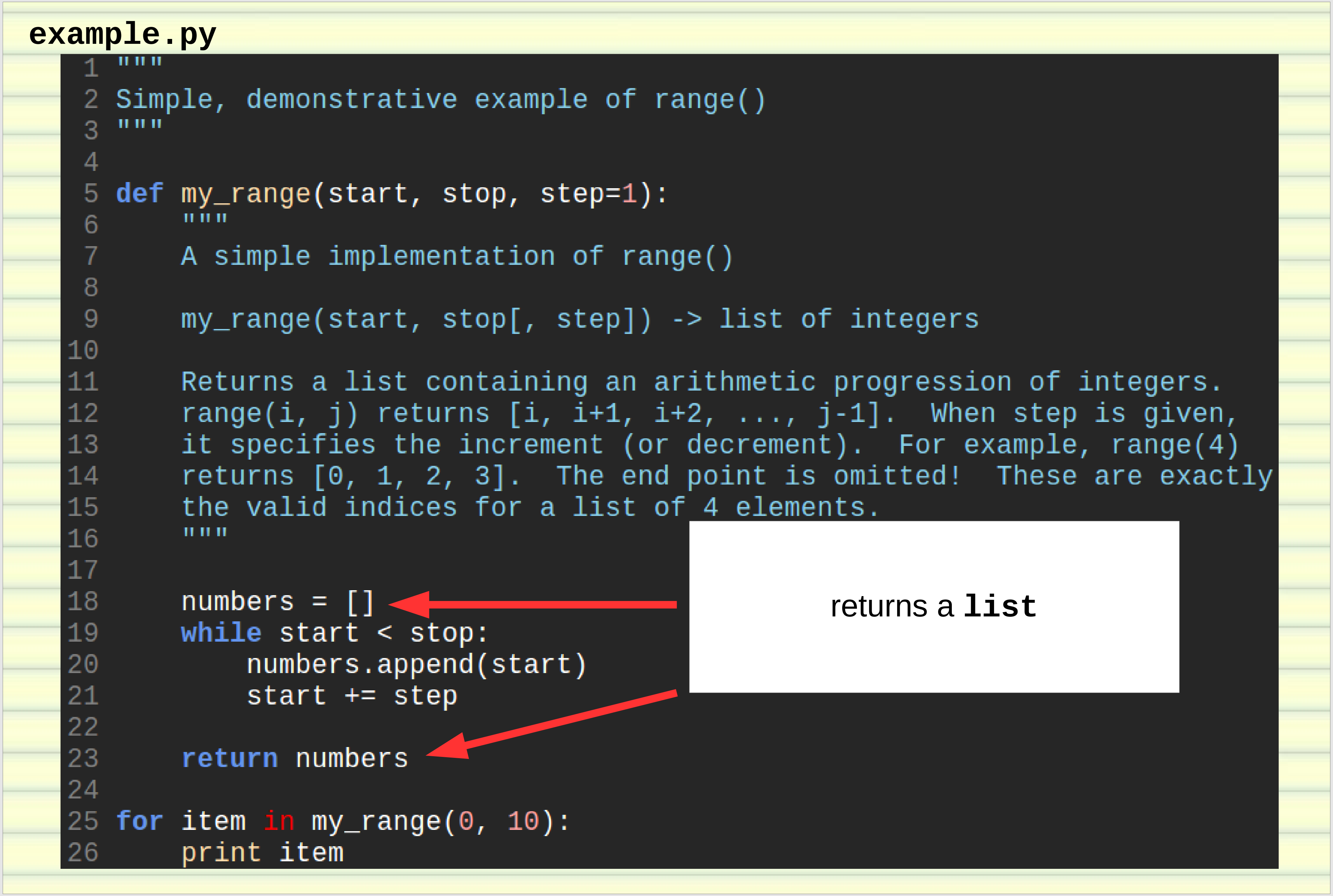


example.py

```
"""
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

    Returns a list contai                                  of integers.
    range(i, j) returns [                                 step is given,
    it specifies the incr                                 mple, range(4)
    returns [0, 1, 2, 3].                                 hese are exactly
    the valid indices for
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

Can be called with or without specifying **step** parameter

Using Functions

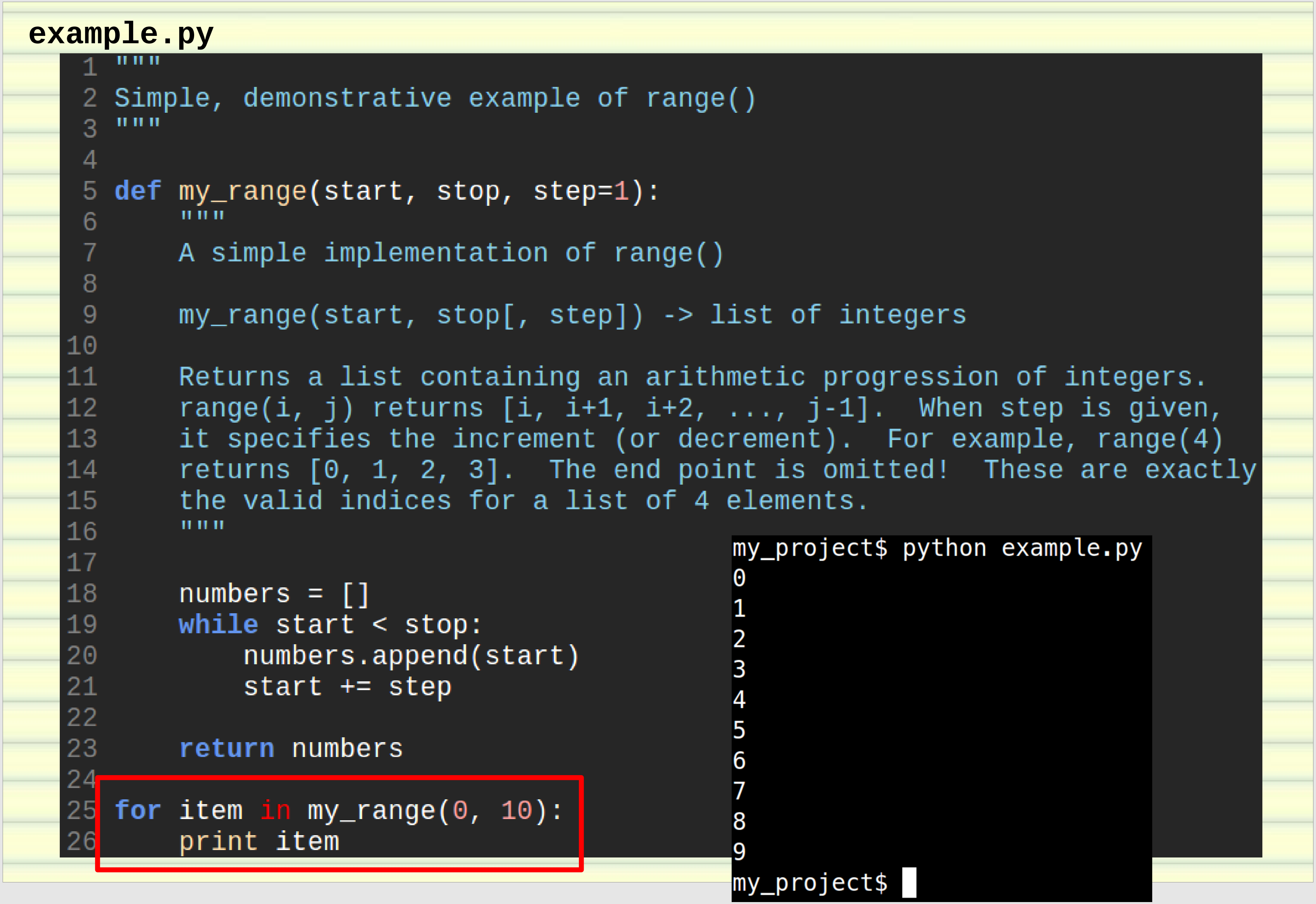


example.py

```python
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

returns a `list`

Using Functions



example.py

```python
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

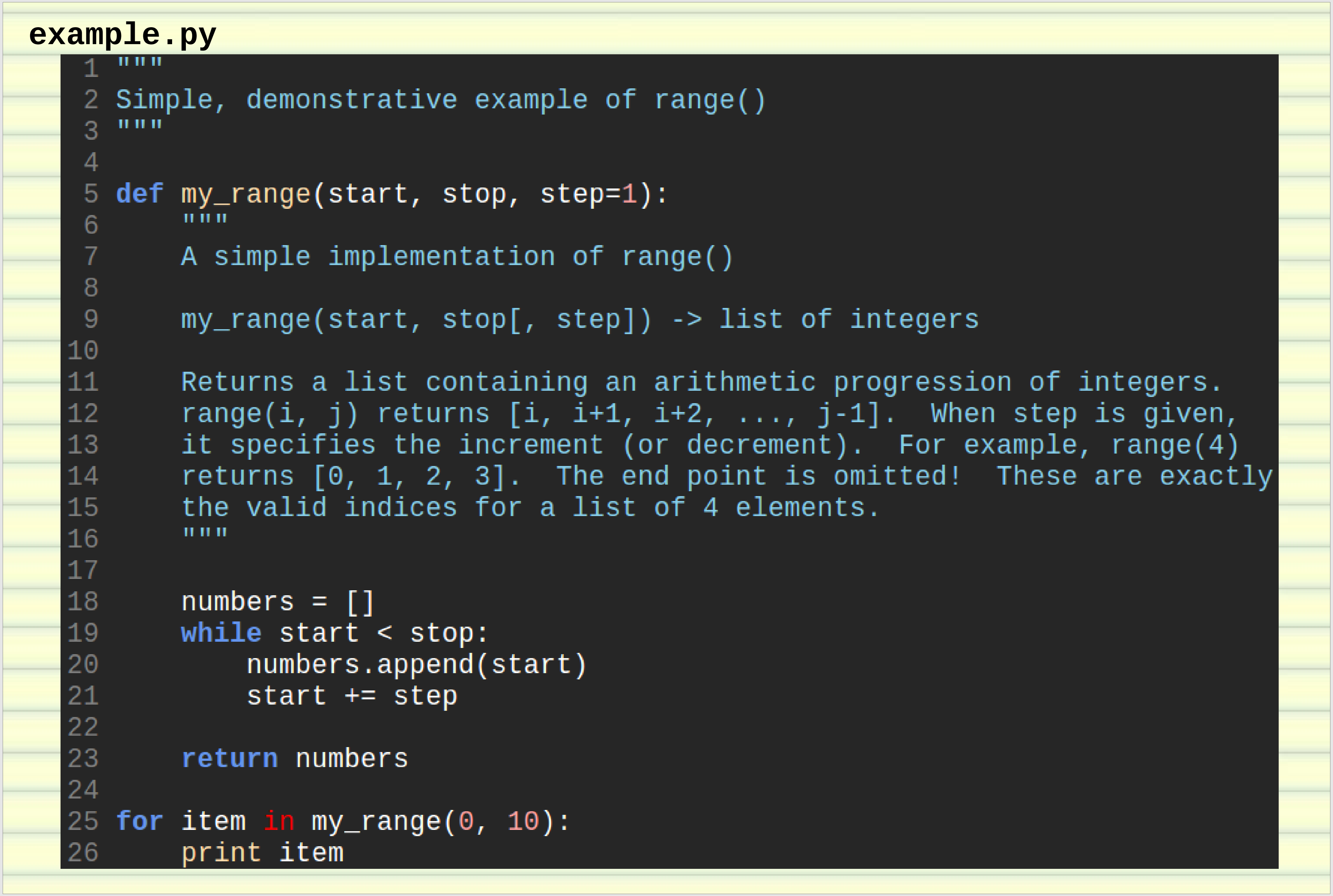
```
my_project$ python example.py
0
1
2
3
4
5
6
7
8
9
my_project$
```

Using Functions -- import

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

```
import
```

Using Functions -- import

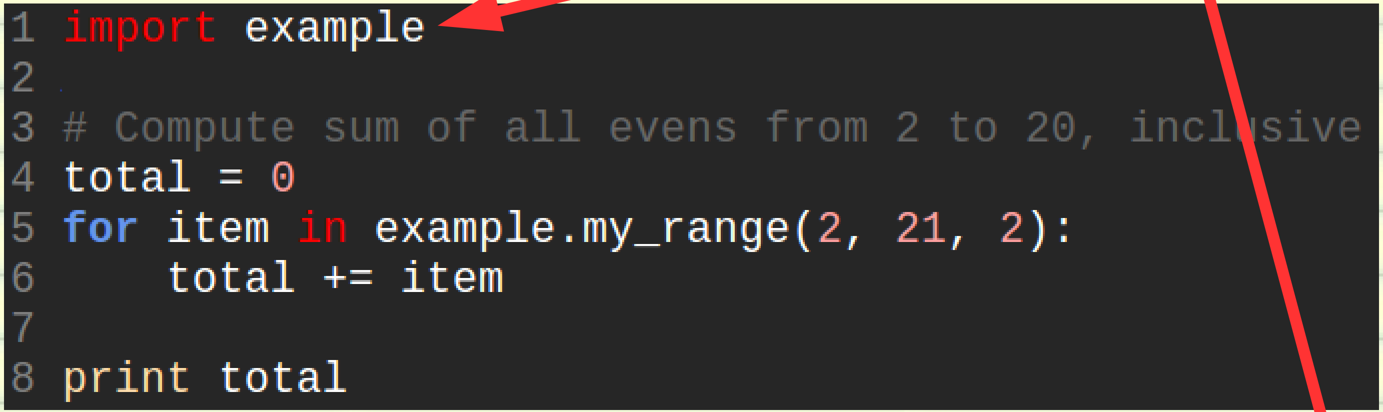


example.py

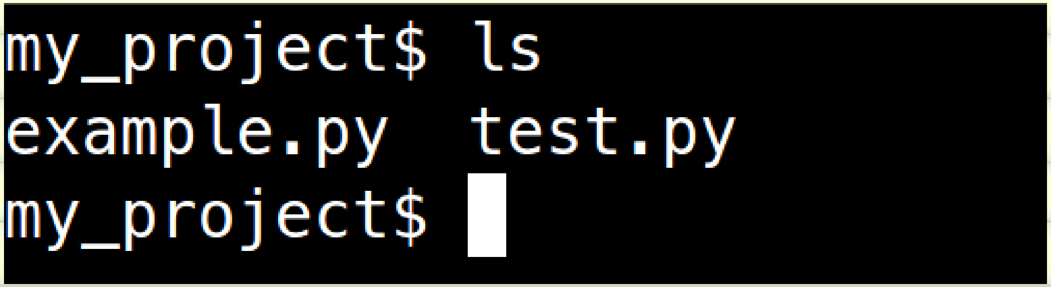
```python
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range()

    my_range(start, stop[, step]) -> list of integers

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

Using Functions -- import

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



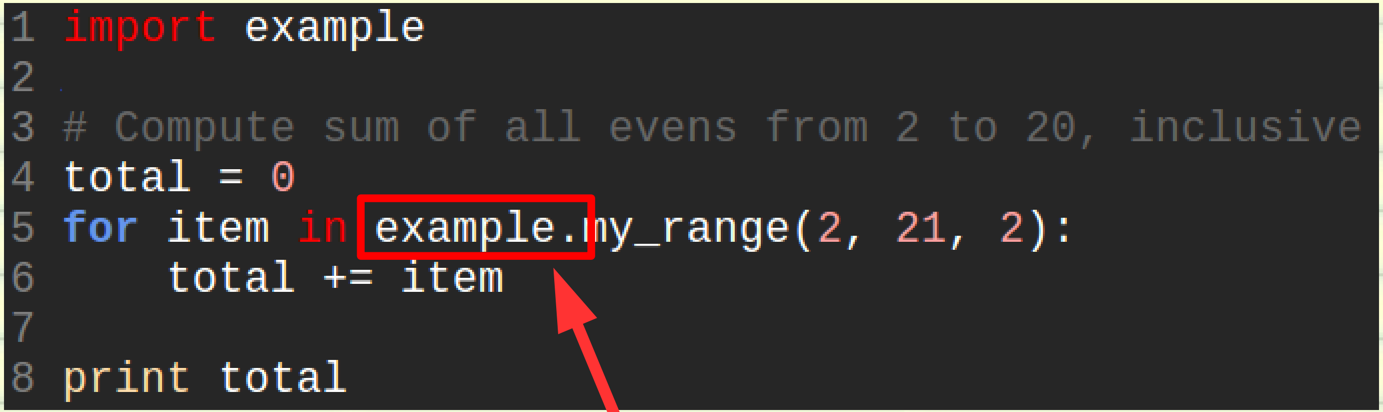
```python
import example

# Compute sum of all evens from 2 to 20, inclusive
total = 0
for item in example.my_range(2, 21, 2):
    total += item

print total
```



```
my_project$ ls
example.py  test.py
my_project$ 
```

Using Functions -- import

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

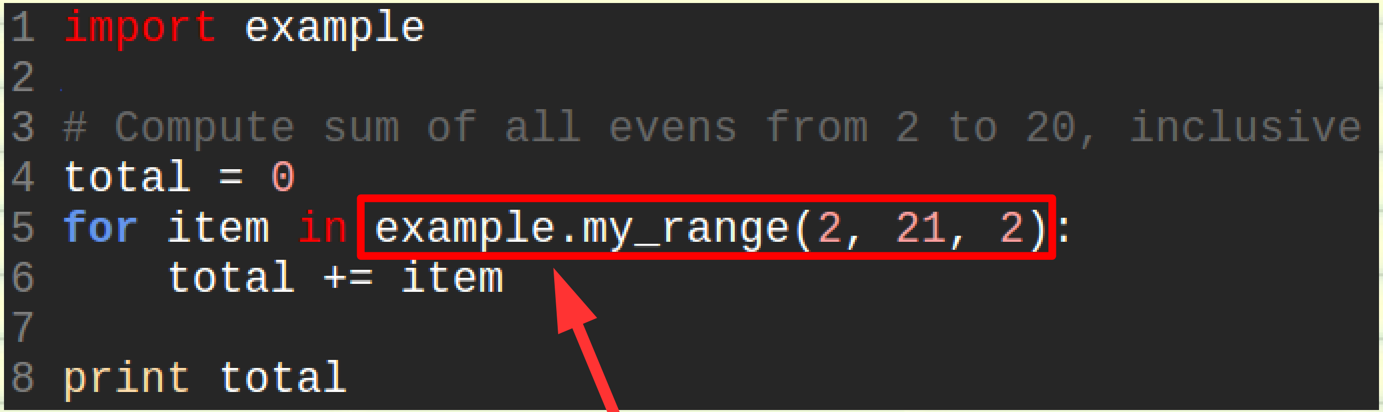


```
import example

# Compute sum of all evens from 2 to 20, inclusive
total = 0
for item in example.my_range(2, 21, 2):
    total += item

print total
```

everything from **example.py** gets loaded into its own ***namespace***

Using Functions -- import

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

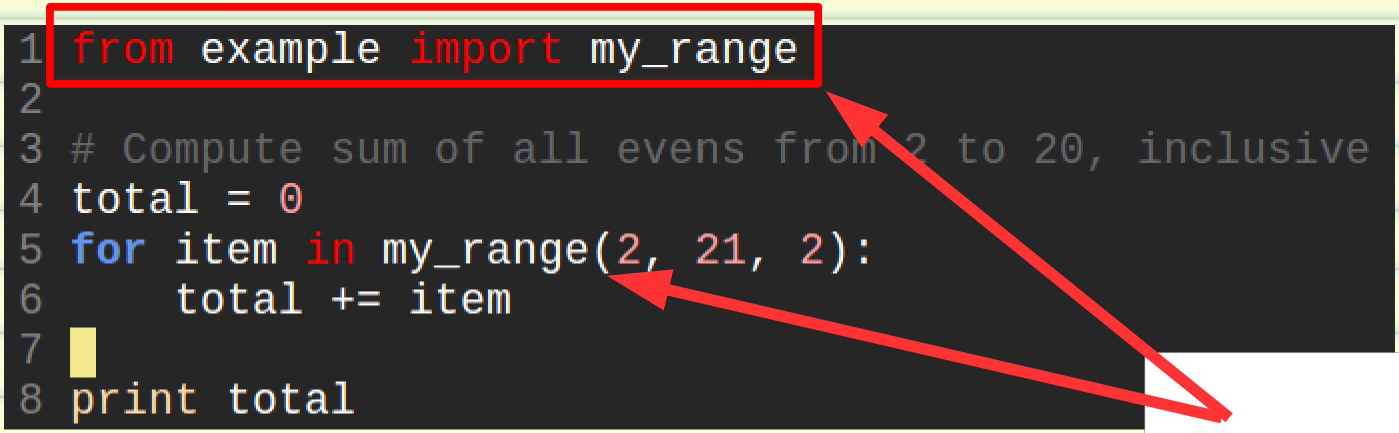


```
import example

# Compute sum of all evens from 2 to 20, inclusive
total = 0
for item in example.my_range(2, 21, 2):
    total += item

print total
```

we access **`my_range()`** from within the **`example`** **namespace** using this notation

Using Functions -- import

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



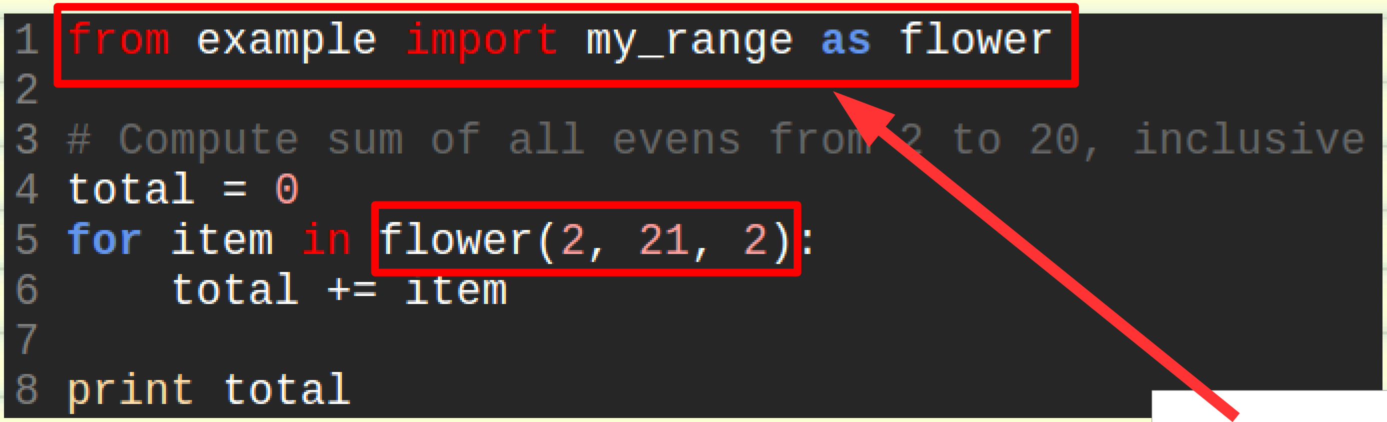
```
from example import my_range

# Compute sum of all evens from 2 to 20, inclusive
total = 0
for item in my_range(2, 21, 2):
    total += item

print total
```

OR

we can import **`my_range`** into the global namespace

Using Functions -- import

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



```
from example import my_range as flower

# Compute sum of all evens from 2 to 20, inclusive
total = 0
for item in flower(2, 21, 2):
    total += item

print total
```

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

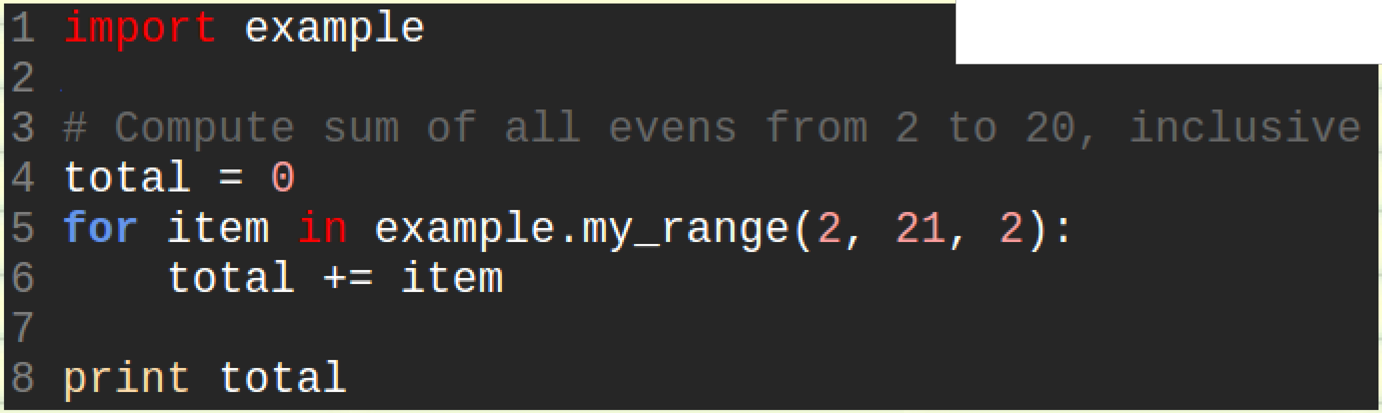
or if we just don't like the name **my_range**

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

Using Functions -- import

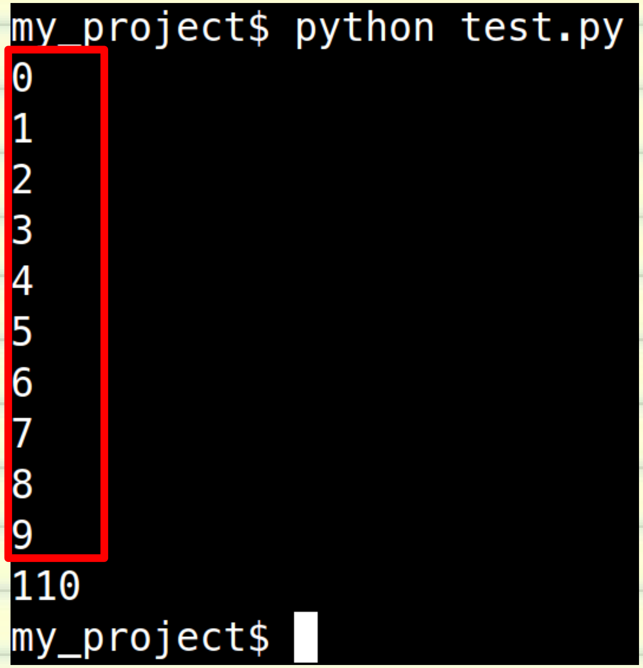
test.py (in the same directory as example.py)

Anyway, so we run this thing.



```
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

print total
```

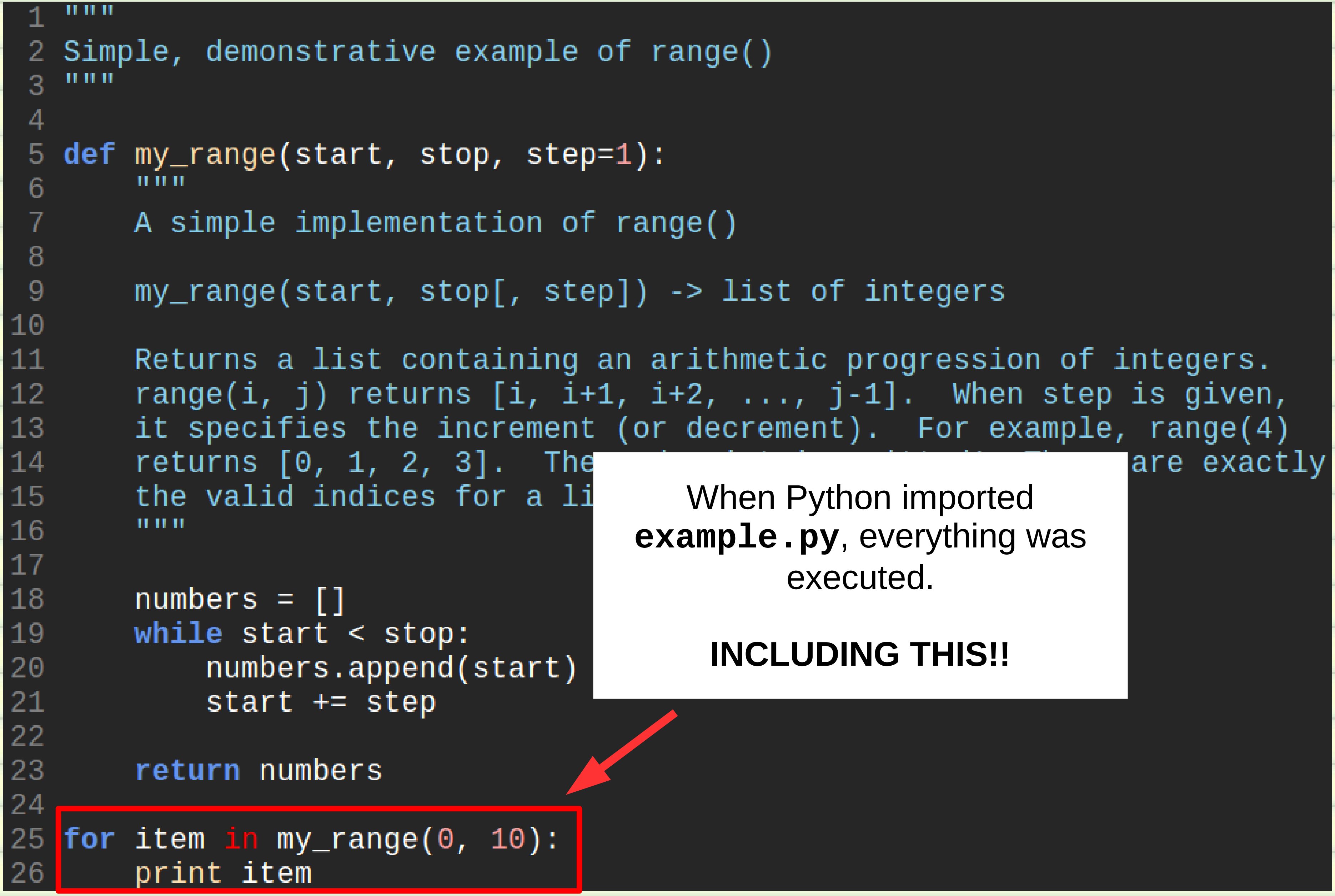
What happened here?!



```
my_project$ python test.py
0
1
2
3
4
5
6
7
8
9
110
my_project$ 
```

Using Functions -- import

example.py



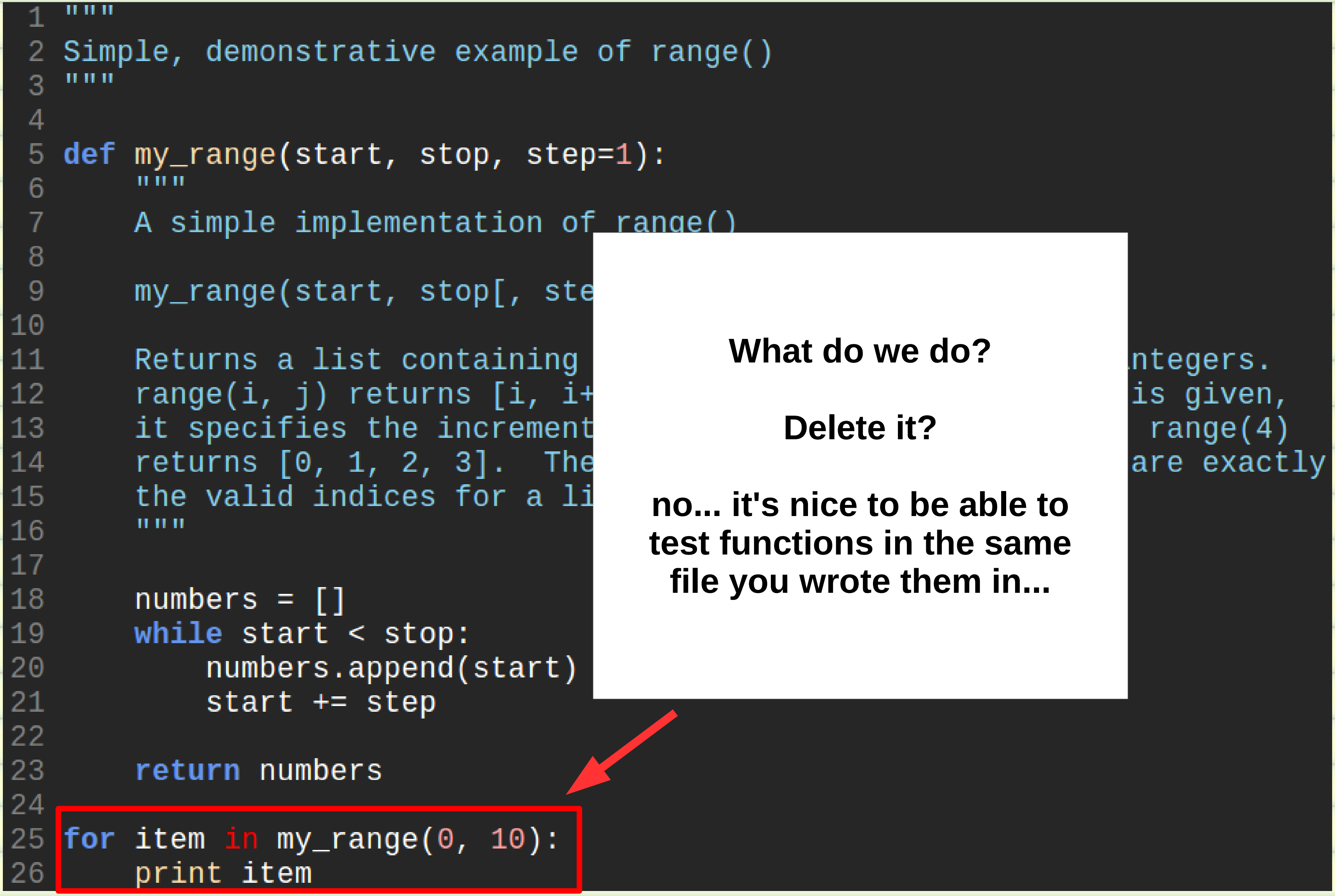
```
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range()

    my_range(start, stop[, step]) -> list of integers

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The                              are exactly
    the valid indices for a li
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

When Python imported **example.py**, everything was executed.

INCLUDING THIS!!

Using Functions -- import

example.py



```
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range()

    my_range(start, stop[, ste

    Returns a list containing                              ntegers.
    range(i, j) returns [i, i+                             is given,
    it specifies the increment                              range(4)
    returns [0, 1, 2, 3].  The                             are exactly
    the valid indices for a li
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

for item in my_range(0, 10):
    print item
```

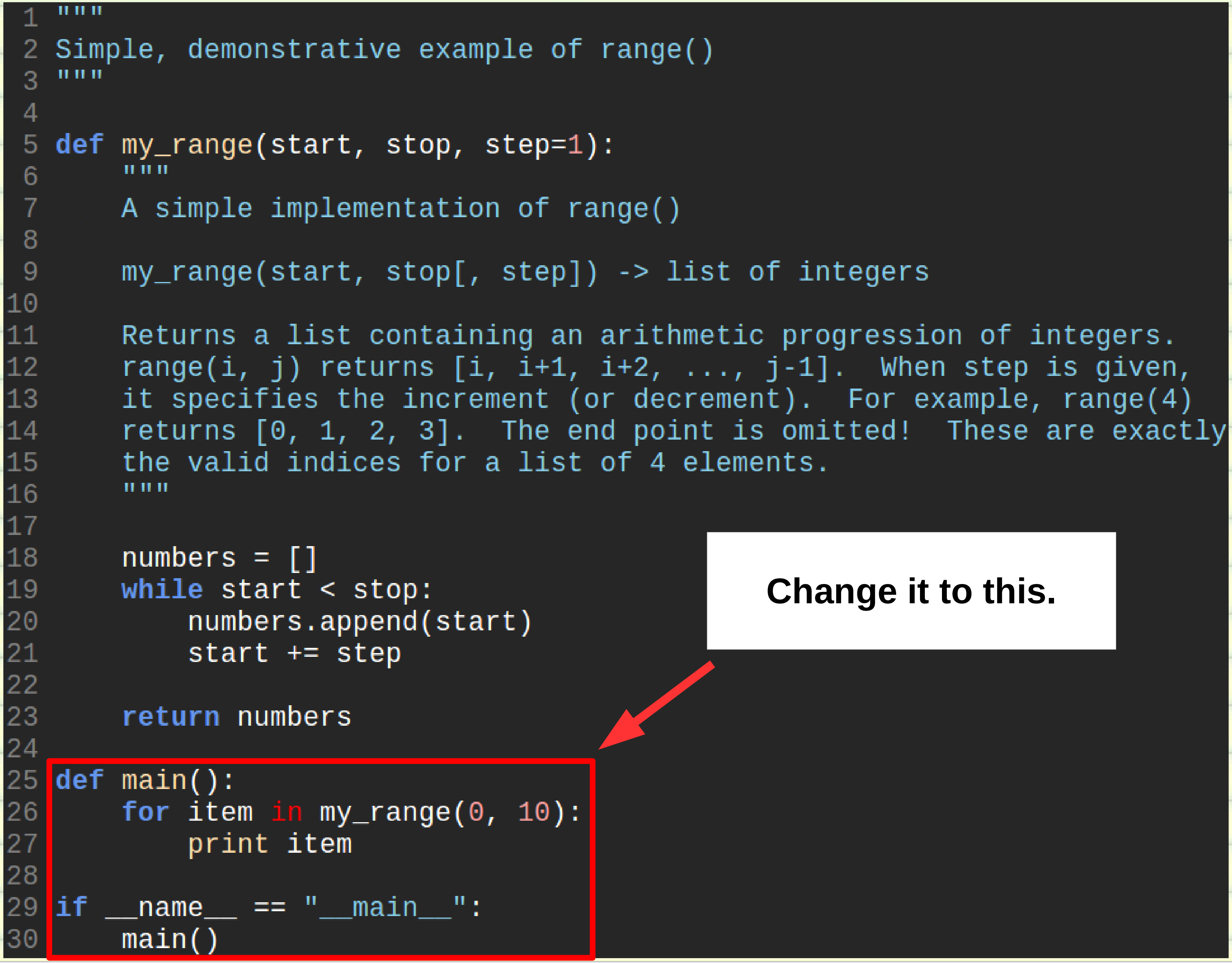
What do we do?

Delete it?

no... it's nice to be able to test functions in the same file you wrote them in...

Using Functions -- import

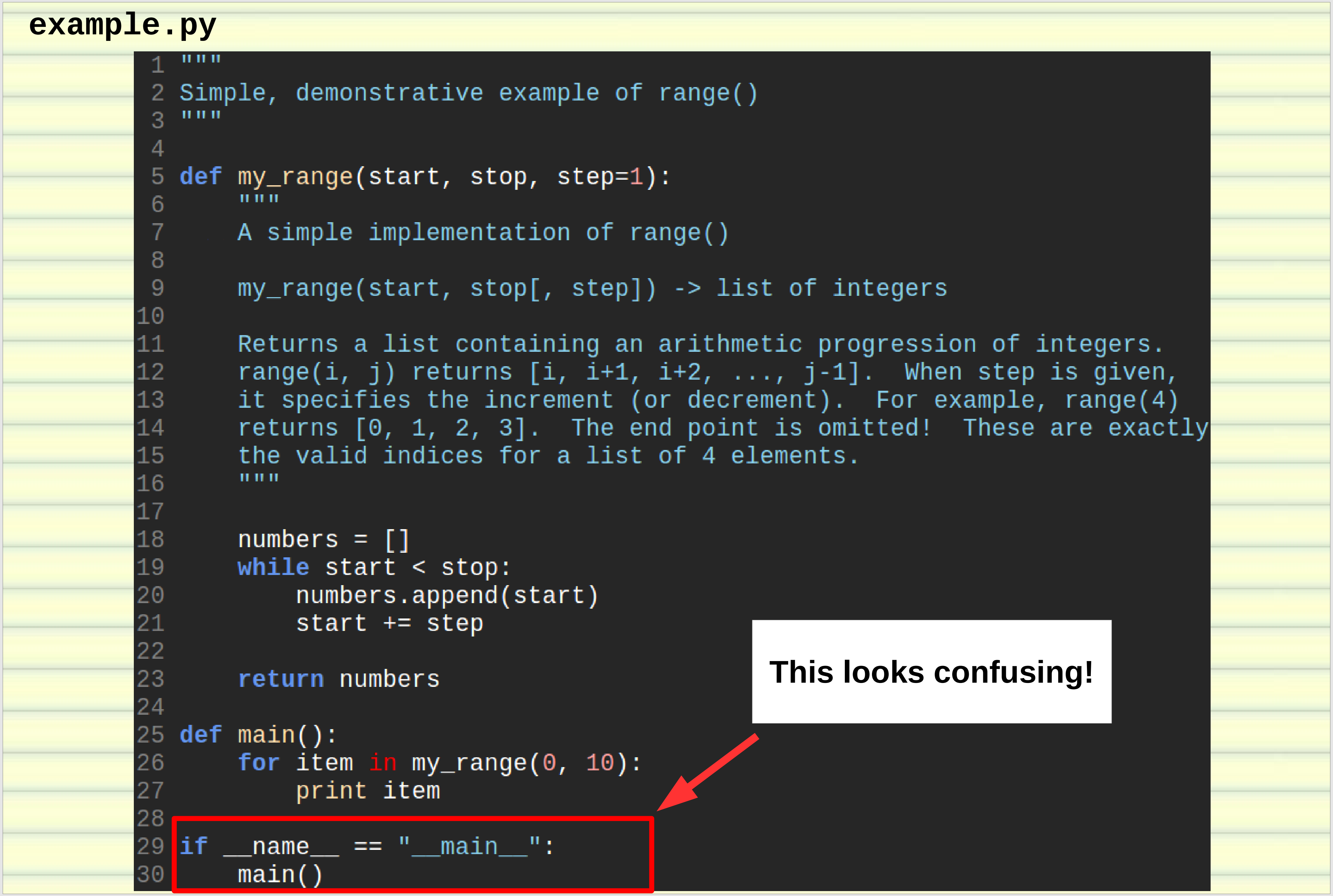
example.py



```
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

Change it to this.

Using Functions -- import



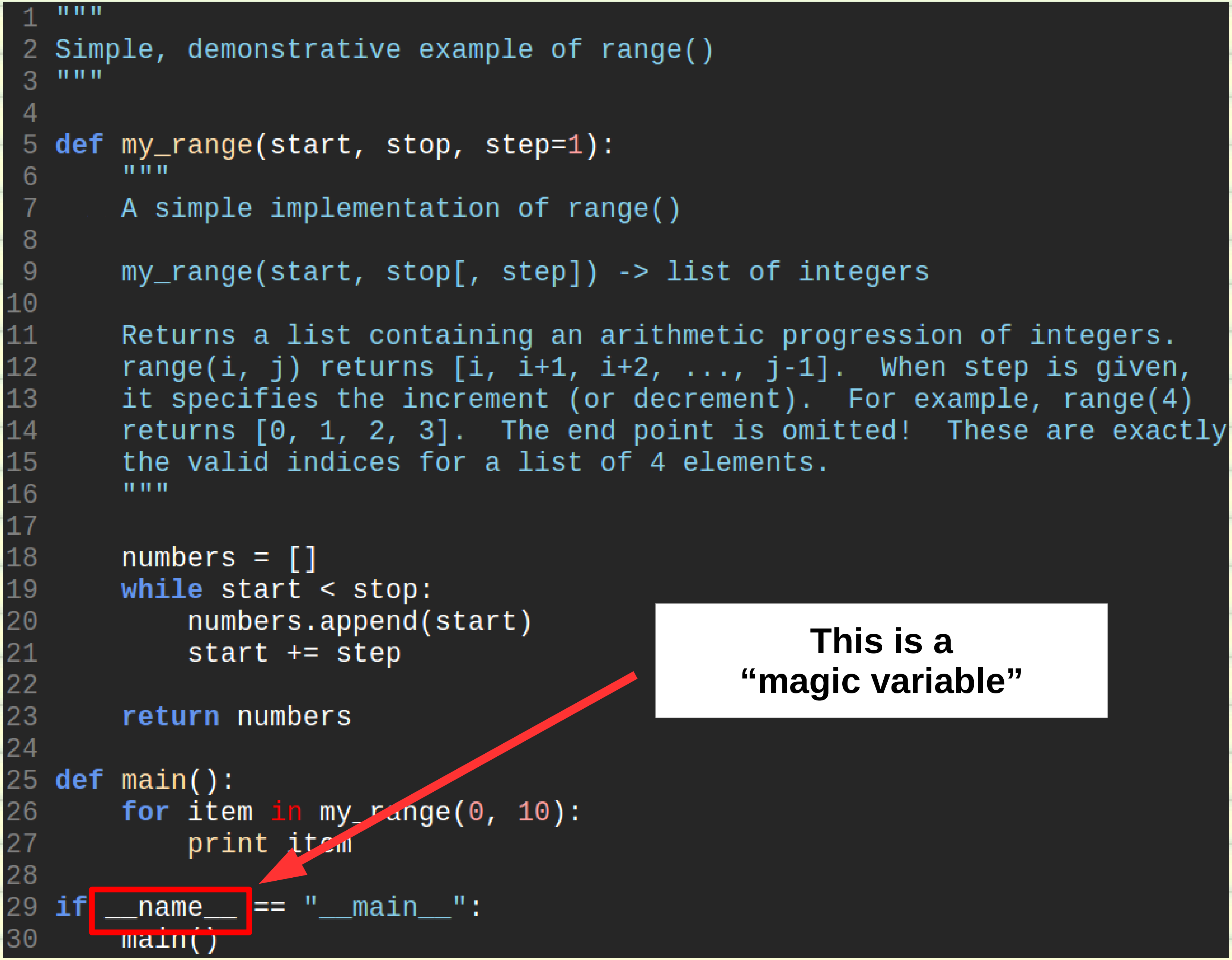
example.py

```python
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

This looks confusing!

Using Functions -- import

example.py

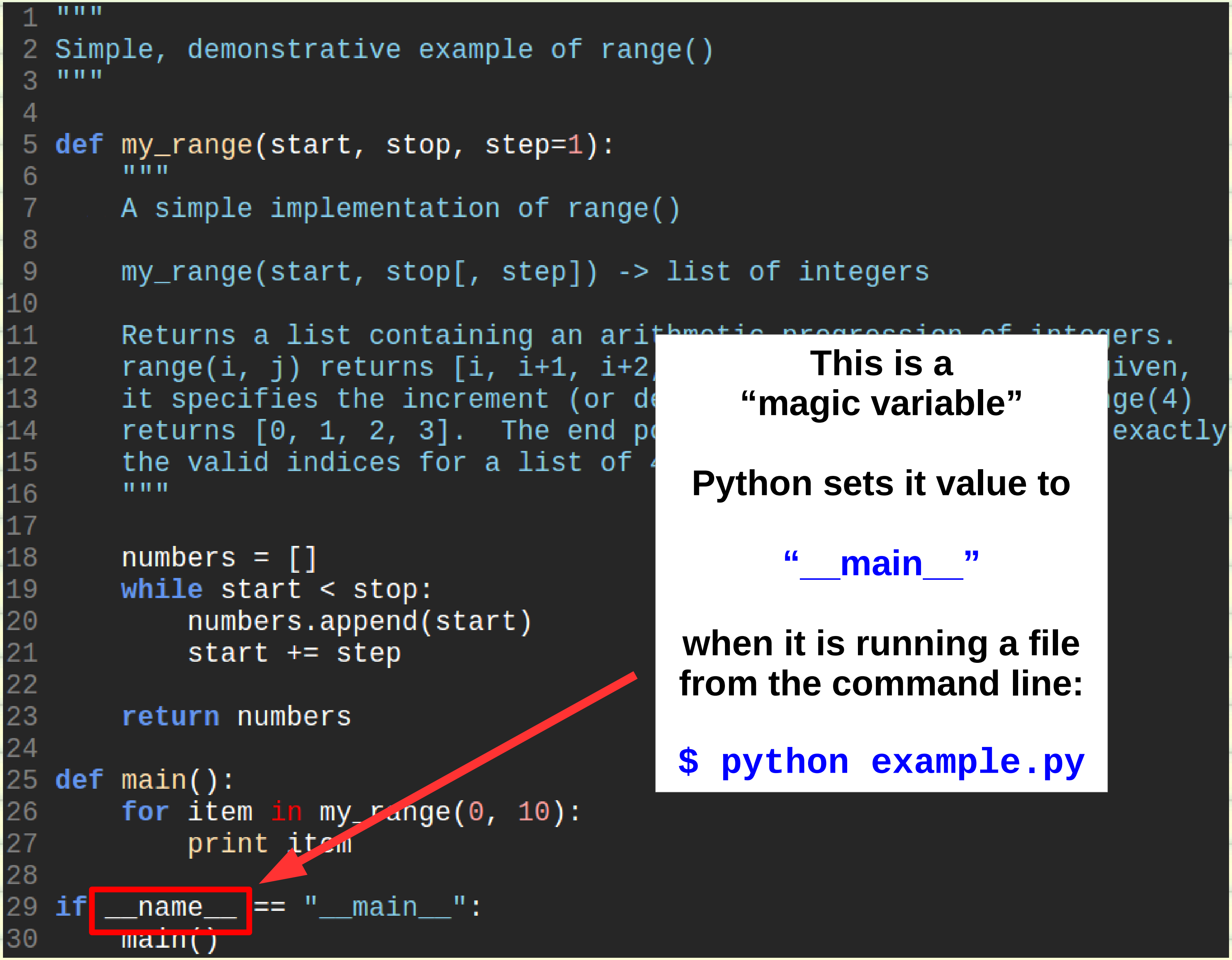


```
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range()

    my_range(start, stop[, step]) -> list of integers

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end point is omitted!  These are exactly
    the valid indices for a list of 4 elements.
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

This is a “magic variable”

Using Functions -- import

example.py



```
"""
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

    Returns a list containing an arithmetic progression of integers.
    range(i, j) returns [i, i+1, i+2,                     iven,
    it specifies the increment (or de                     ge(4)
    returns [0, 1, 2, 3].  The end p                      exactly
    the valid indices for a list of 4
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

This is a "magic variable"

Python sets it value to

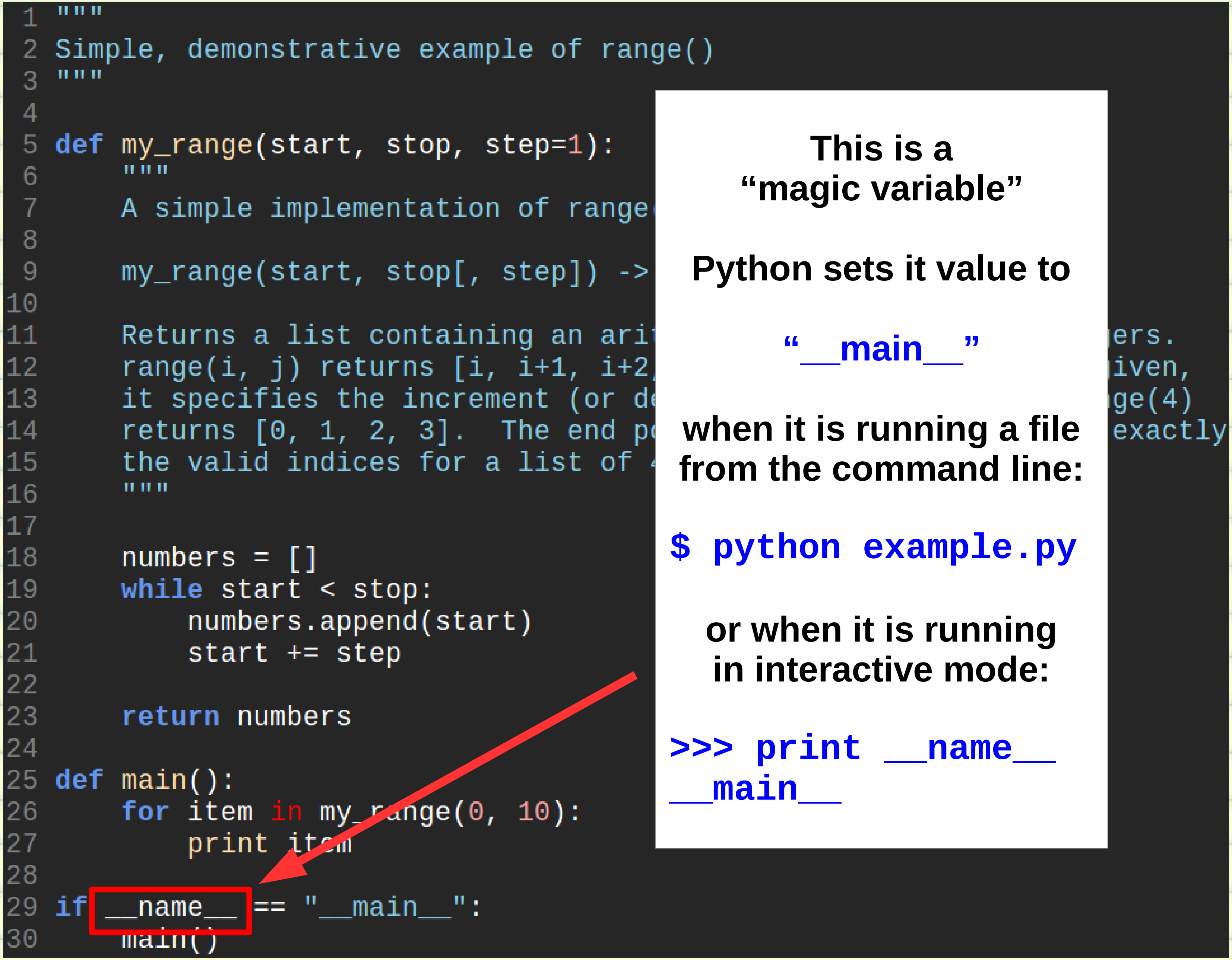
"__main__"

when it is running a file from the command line:

`$ python example.py`

Using Functions -- import

example.py



```
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range(

    my_range(start, stop[, step]) ->

    Returns a list containing an arit                           ers.
    range(i, j) returns [i, i+1, i+2,                           iven,
    it specifies the increment (or de                           ge(4)
    returns [0, 1, 2, 3].  The end po                           exactly
    the valid indices for a list of 4
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

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:

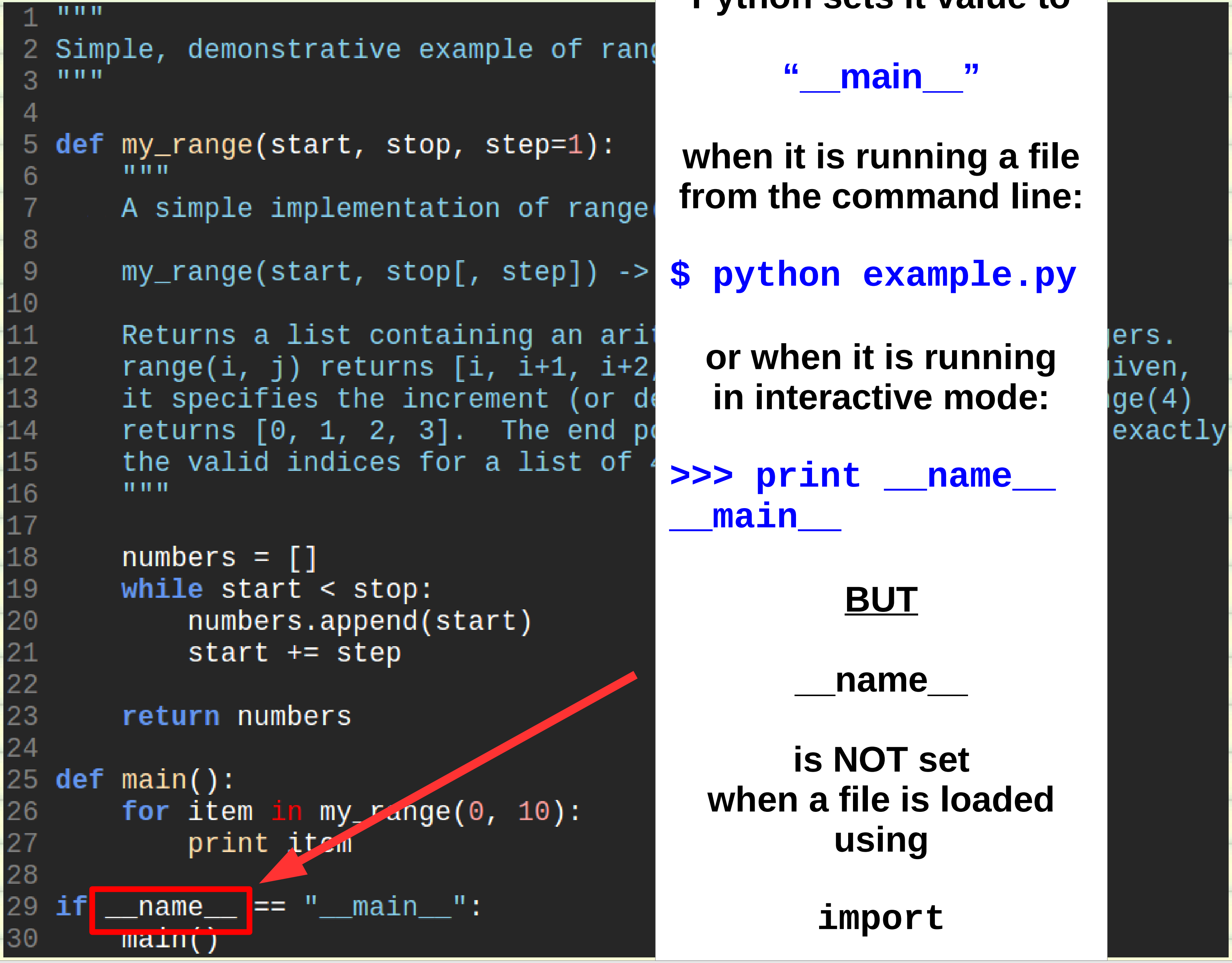
```
$ python example.py
```

or when it is running in interactive mode:

```
>>> print __name__
__main__
```

Using Functions -- import

example.py



```
"""
Simple, demonstrative example of rang
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range(

    my_range(start, stop[, step]) ->

    Returns a list containing an arit                       ers.
    range(i, j) returns [i, i+1, i+2,                       iven,
    it specifies the increment (or de                       ge(4)
    returns [0, 1, 2, 3].  The end po                       exactly
    the valid indices for a list of 4
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

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__
```

BUT

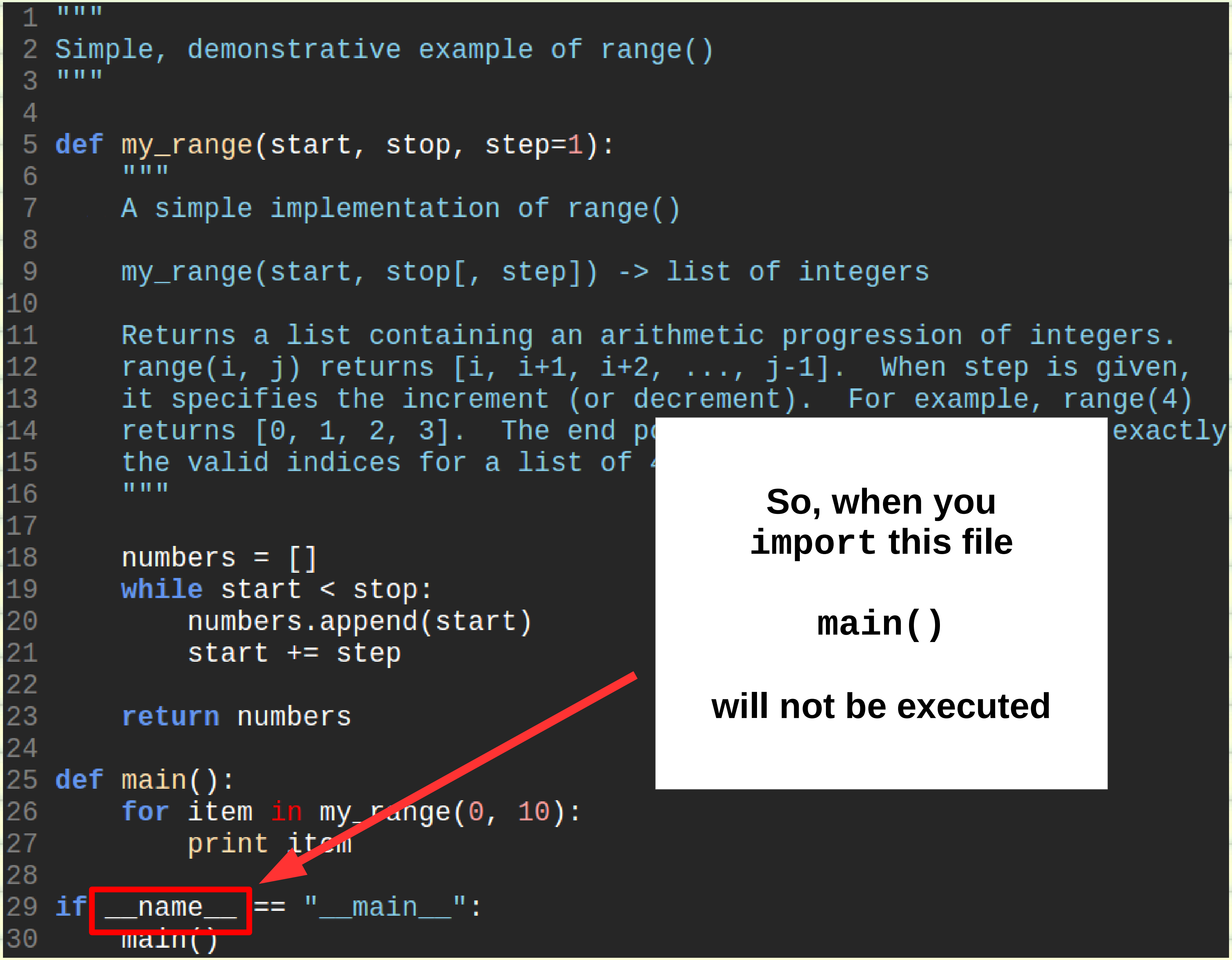
__name__

is NOT set when a file is loaded using

`import`

Using Functions -- import

example.py



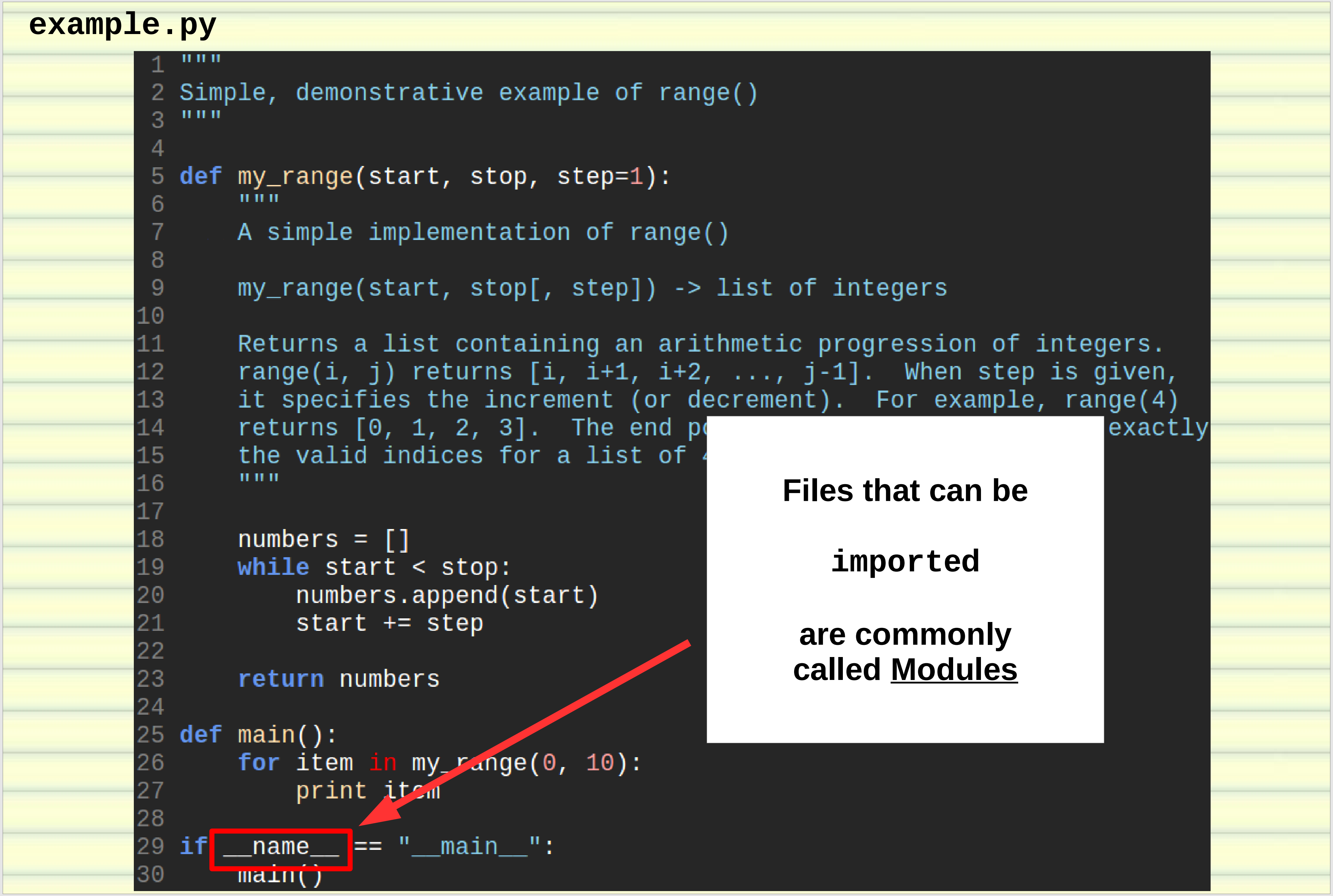
```
"""
Simple, demonstrative example of range()
"""

def my_range(start, stop, step=1):
    """
    A simple implementation of range()

    my_range(start, stop[, step]) -> list of integers

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end po                          exactly
    the valid indices for a list of 4
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

So, when you `import` this file

`main()`

will not be executed

Using Functions -- import



example.py

```
"""
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

    Returns a list containing an arithmetic progression of integers.
    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)
    returns [0, 1, 2, 3].  The end po                          exactly
    the valid indices for a list of
    """

    numbers = []
    while start < stop:
        numbers.append(start)
        start += step

    return numbers

def main():
    for item in my_range(0, 10):
        print item

if __name__ == "__main__":
    main()
```

Files that can be

`imported`

are commonly called <u>Modules</u>

Using Functions -- import

Questions?