61A Lecture 11

Friday, September 23

A Function with Behavior That Varies Over Time

```
Let's model a bank account that has a balance of $100
```

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
```

Argument:
amount to withdraw

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value:
remaining balance

```
>>> withdraw(25)
75
```

Argument:
amount to withdraw

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value:
remaining balance

```
>>> withdraw(25)
75
```

Argument:
amount to withdraw

```
>>> withdraw(25)
```

Second withdrawal
of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

```
>>> withdraw(25)
75
```

Argument: amount to withdraw

Different return value!

```
>>> withdraw(25)
50
```

Second withdrawal of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value:
remaining balance

Different
return value!

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
```

Argument:
amount to withdraw

Second withdrawal
of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'
```

Different return value!

Second withdrawal of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

Return value: remaining balance

Argument: amount to withdraw

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

Different return value!

Second withdrawal of the same amount

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

Where's this balance stored?

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

Second withdrawal of the same amount

Where's this balance stored?

```
>>> withdraw = make_withdraw(100)
```

A Function with Behavior That Varies Over Time

Let's model a bank account that has a balance of $100

```
>>> withdraw(25)
75

>>> withdraw(25)
50

>>> withdraw(60)
'Insufficient funds'

>>> withdraw(15)
35
```

Return value: remaining balance

Argument: amount to withdraw

Different return value!

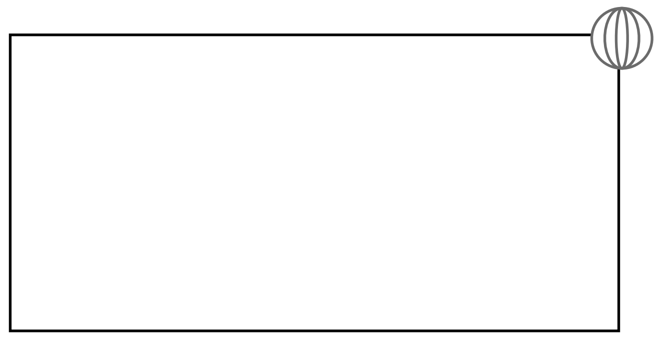
Second withdrawal of the same amount

Where's this balance stored?

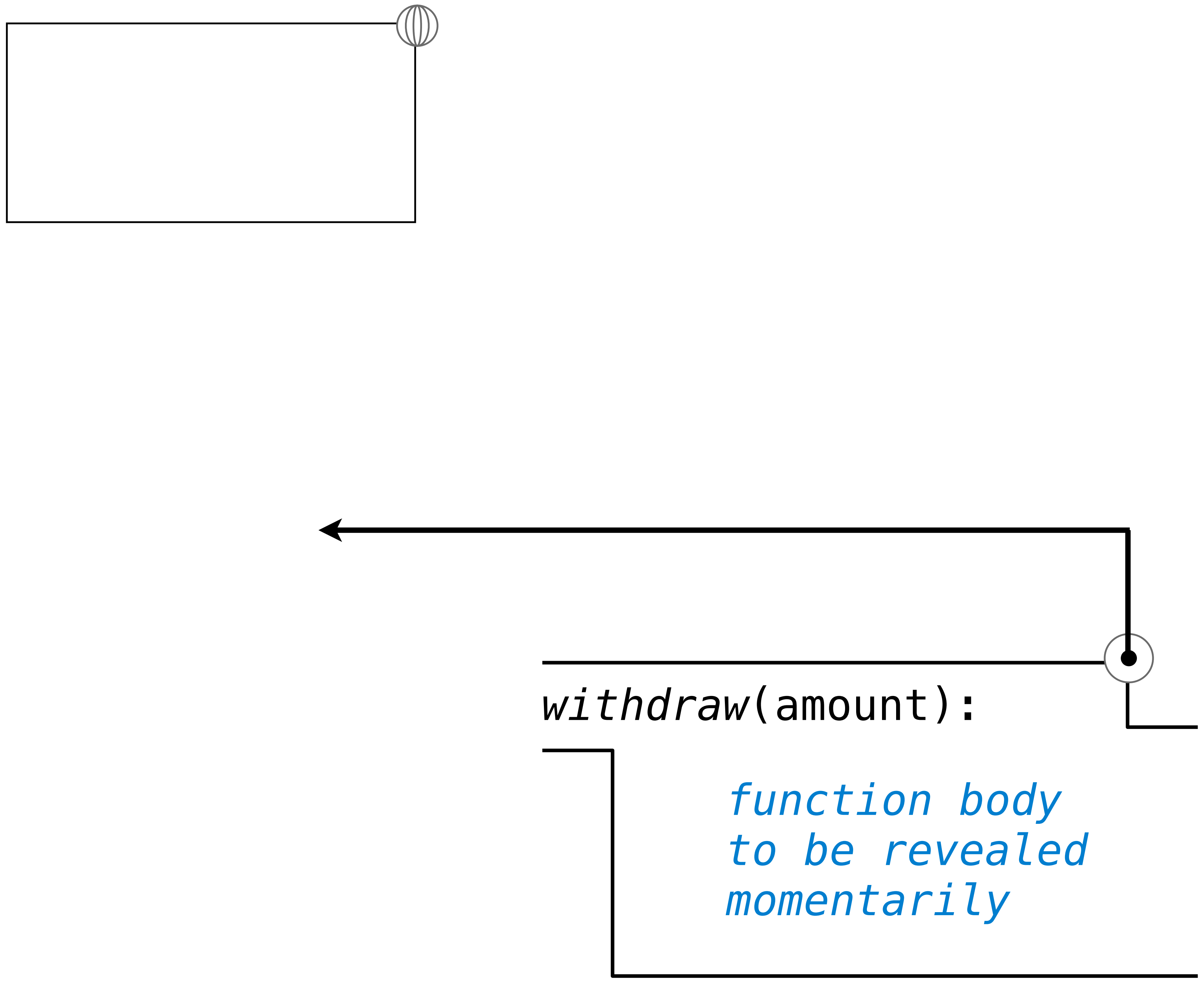
```
>>> withdraw = make_withdraw(100)
```

Within the function!

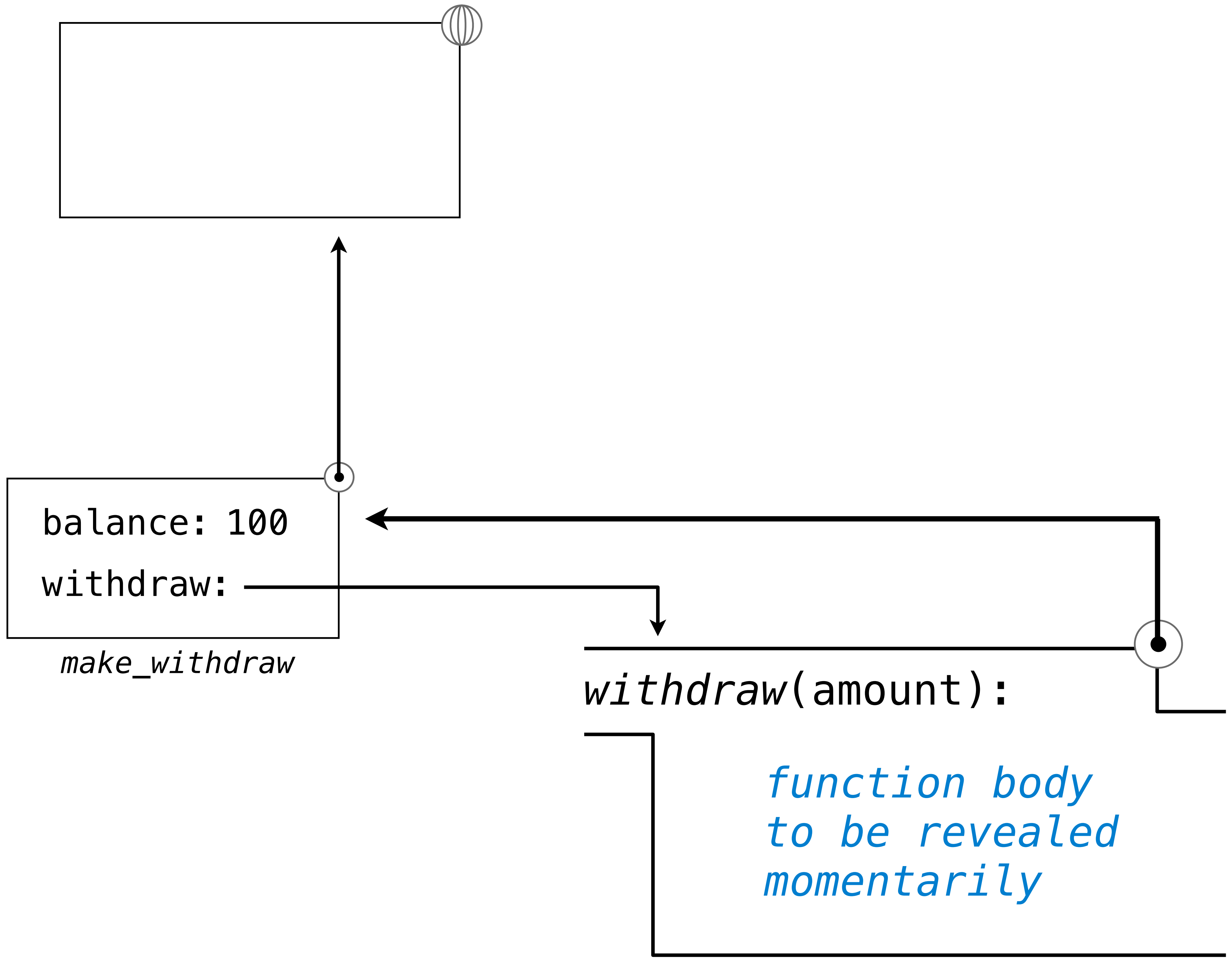
Persistent Local State



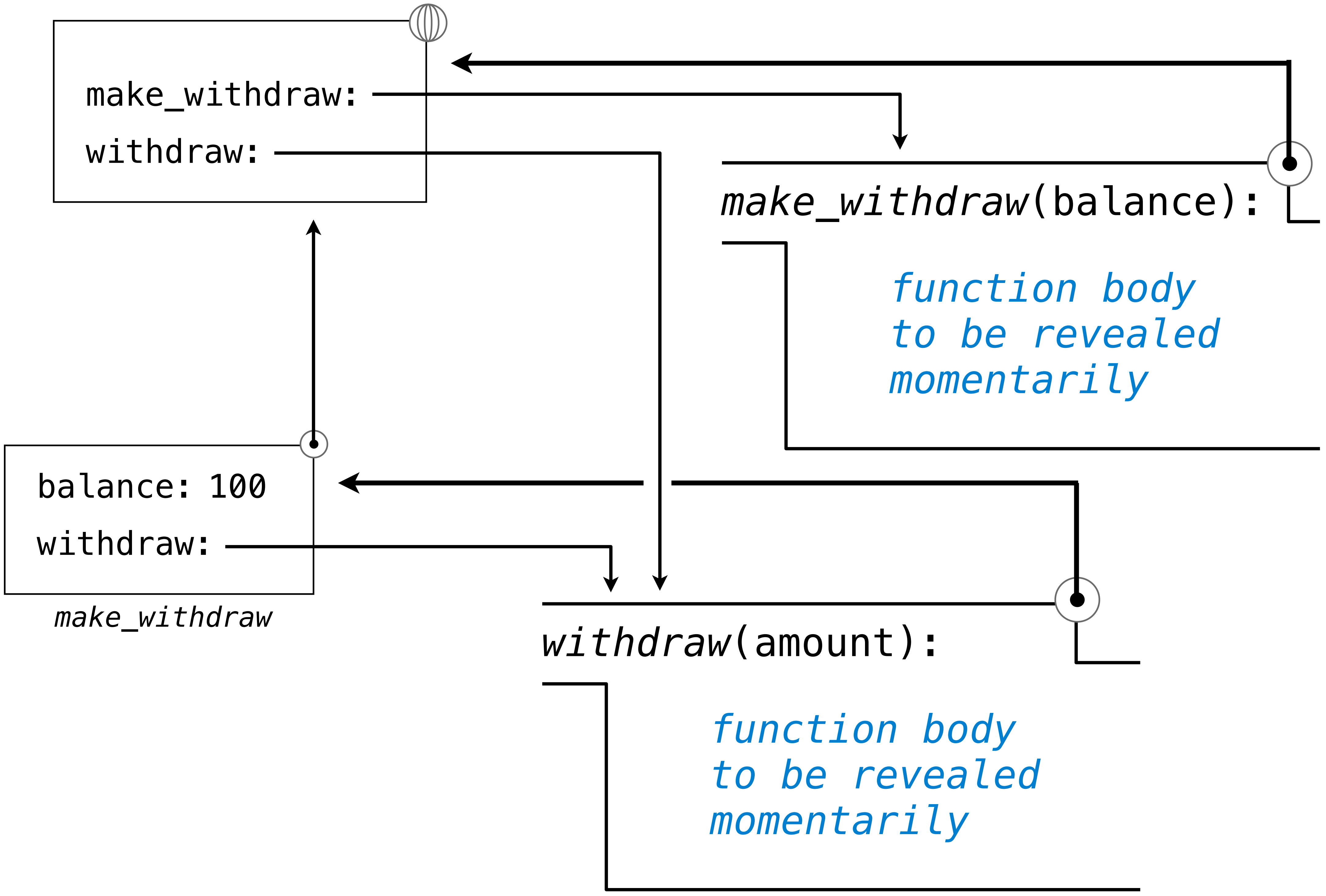
Persistent Local State



Persistent Local State



Persistent Local State



Local State via Non-Local Assignment

Local State via Non-Local Assignment

```
def make_withdraw(balance):
```

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
```

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
```

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
```

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
```

Declare the name "balance" nonlocal

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
```

Declare the name "balance" nonlocal

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
```

Declare the name "balance" nonlocal

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
```

Declare the name "balance" nonlocal

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
```

Declare the name "balance" nonlocal

Re-bind balance where it was bound previously

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
```

Declare the name "balance" nonlocal

Re-bind balance where it was bound previously

Local State via Non-Local Assignment

```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Declare the name "balance" nonlocal

Re-bind balance where it was bound previously

Local State via Non-Local Assignment

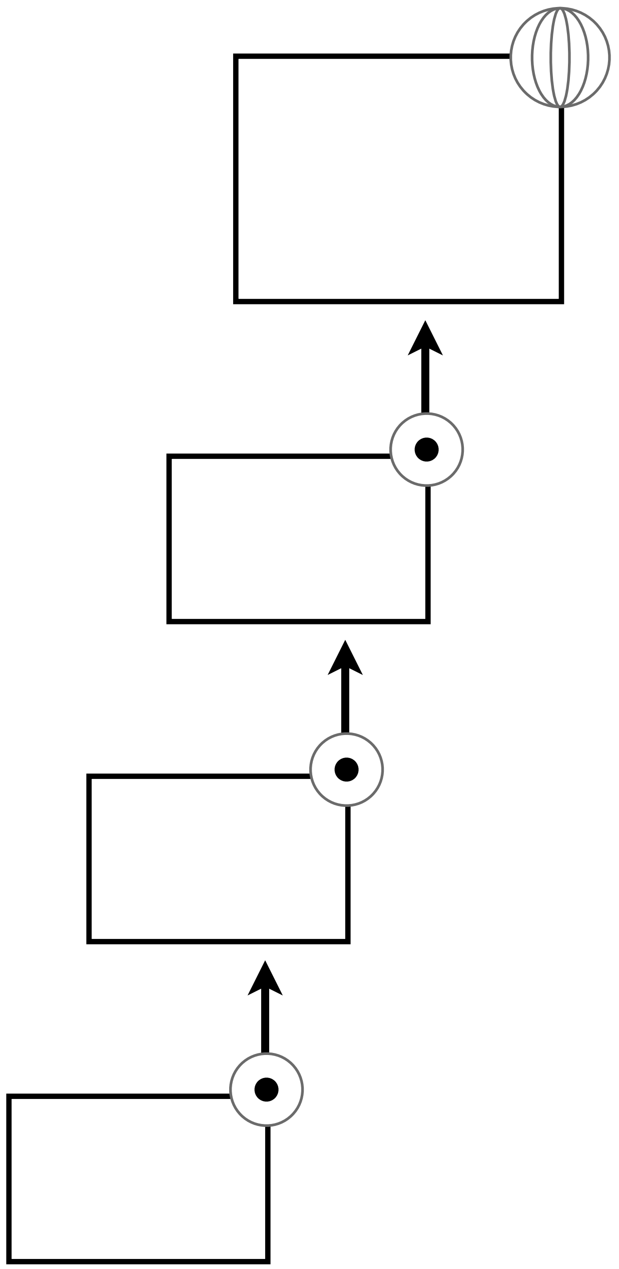
```
def make_withdraw(balance):
    """Return a withdraw function with a starting balance."""
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

Declare the name "balance" nonlocal

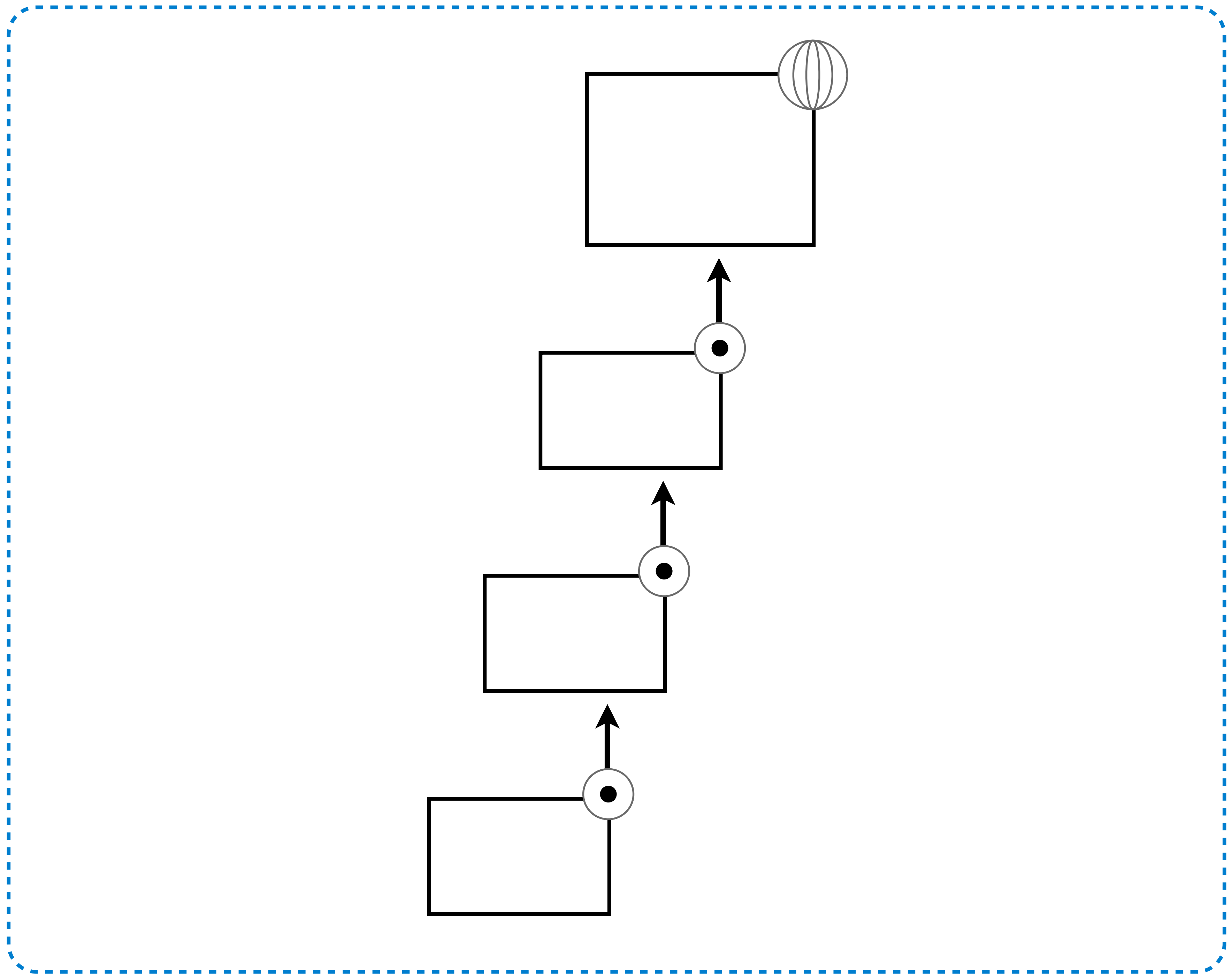
Re-bind balance where it was bound previously

Demo

Local, Non-Local, and Global Frames

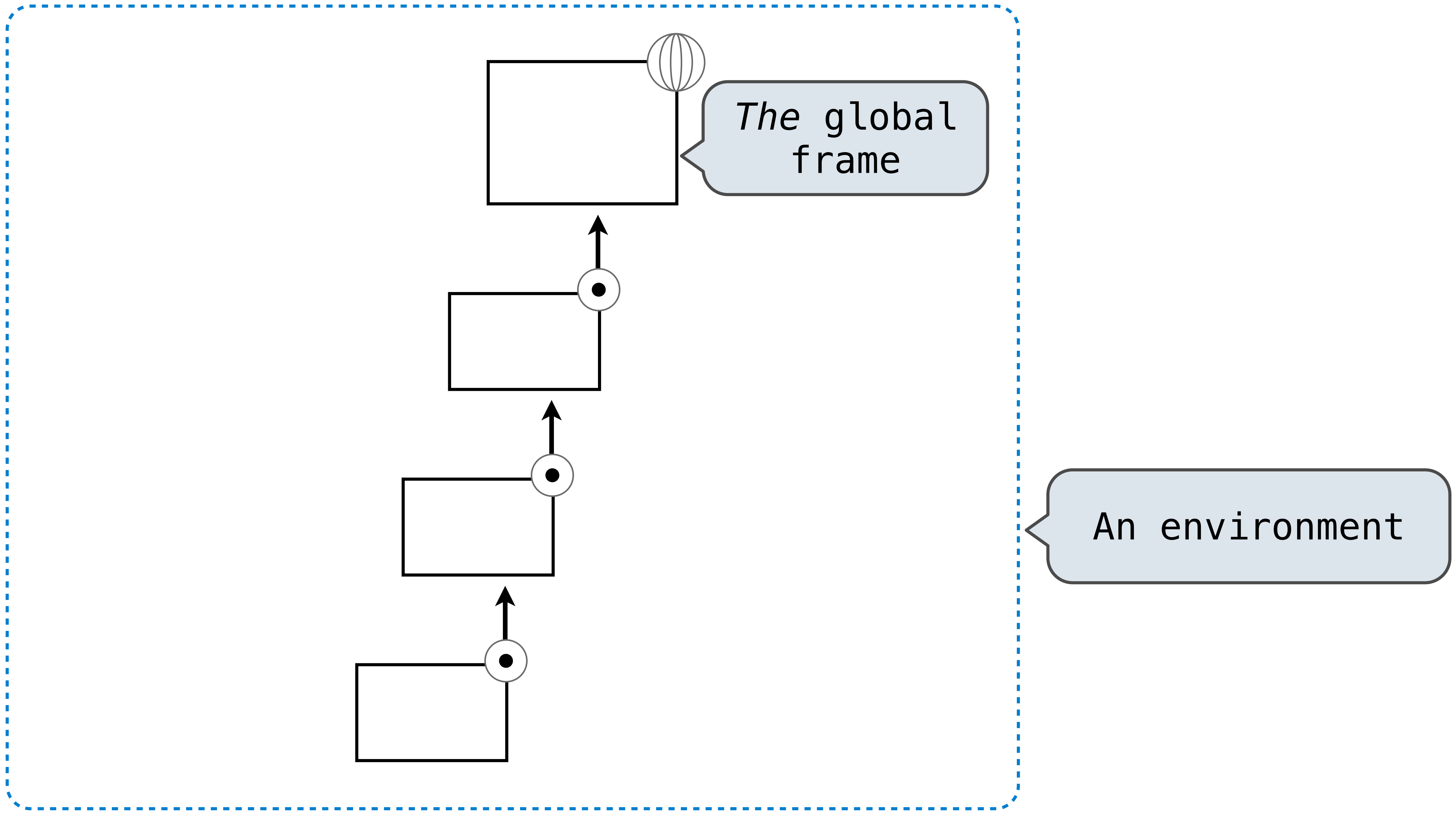


Local, Non-Local, and Global Frames

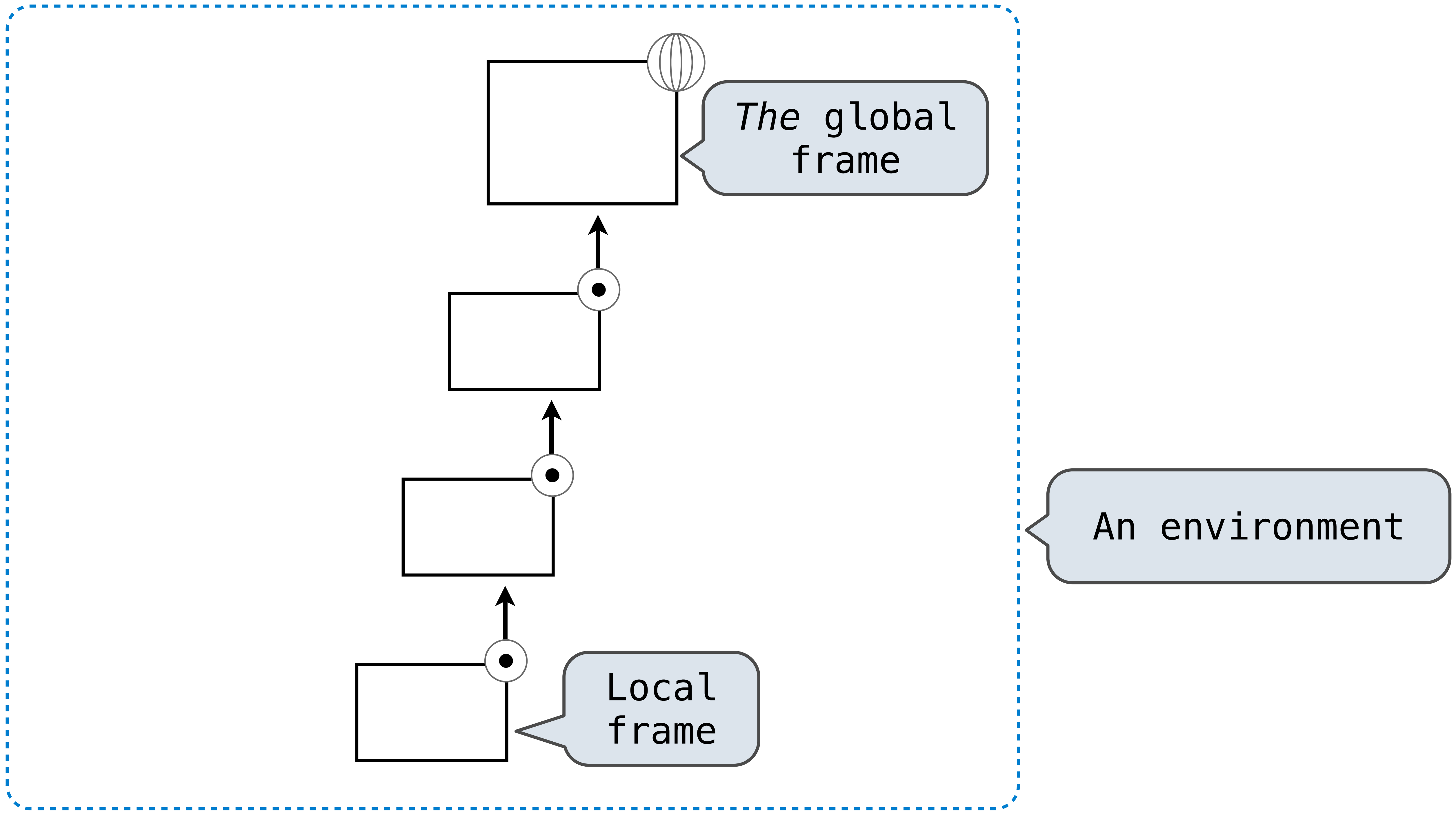


An environment

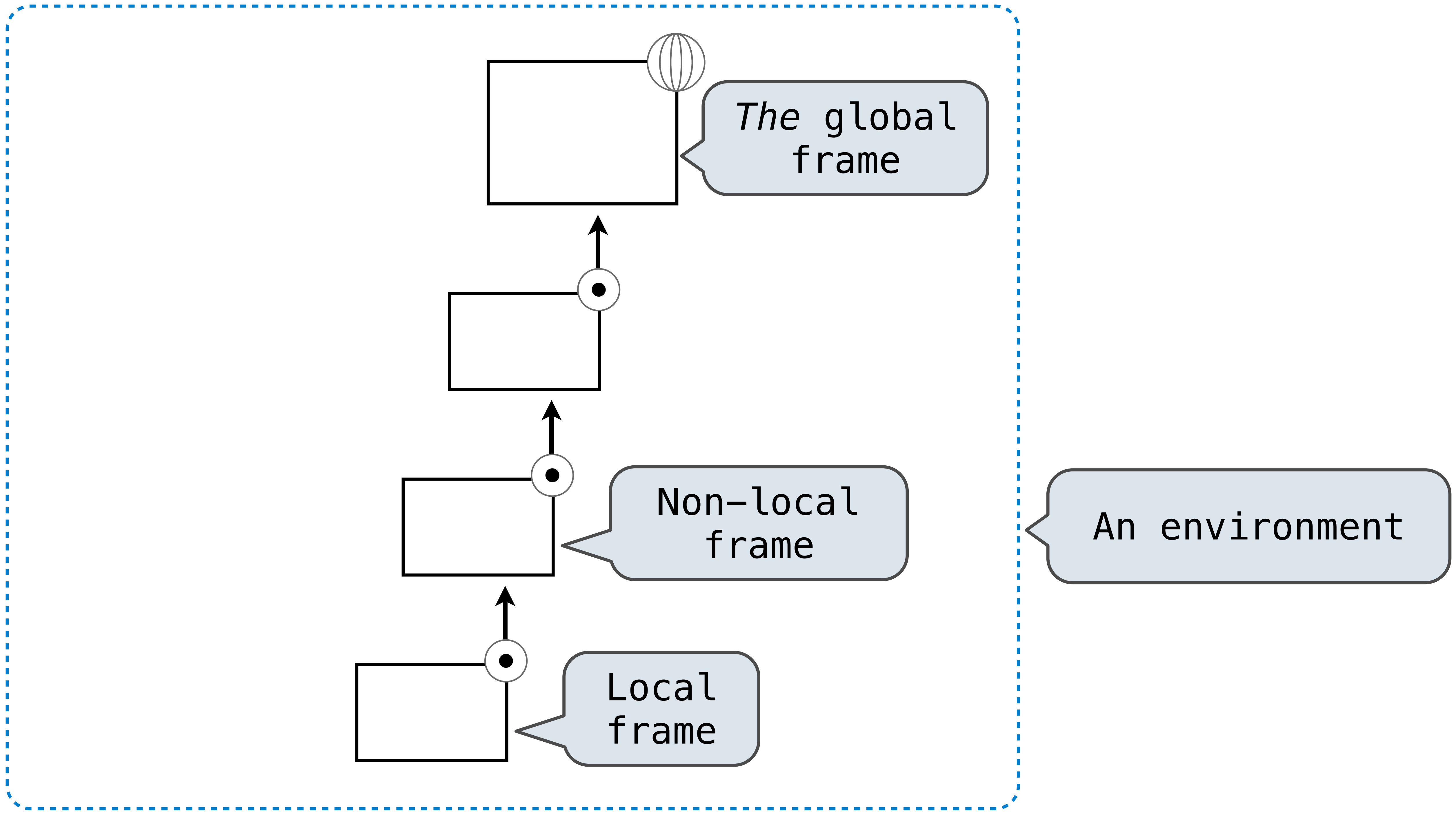
Local, Non-Local, and Global Frames



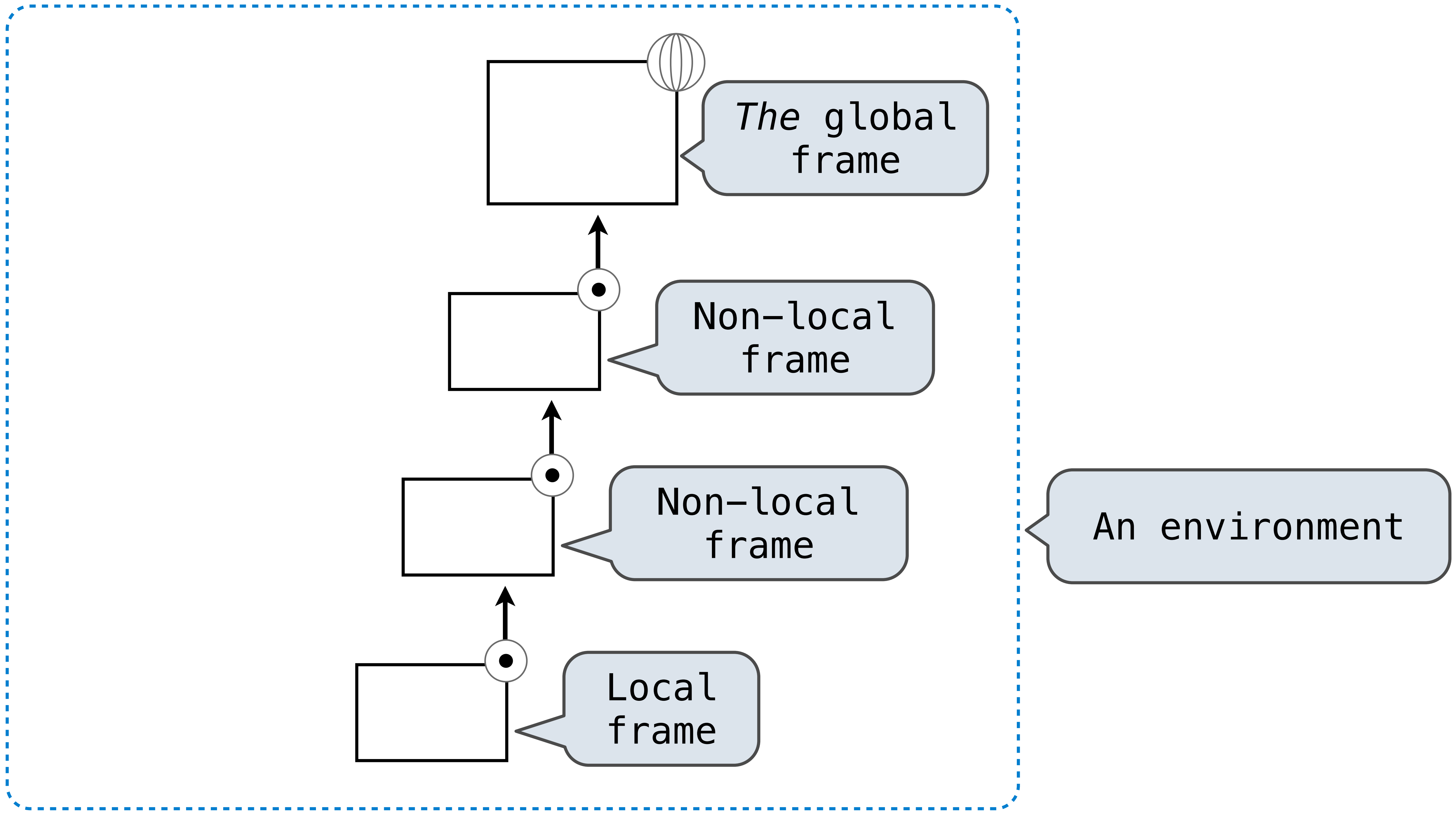
Local, Non-Local, and Global Frames



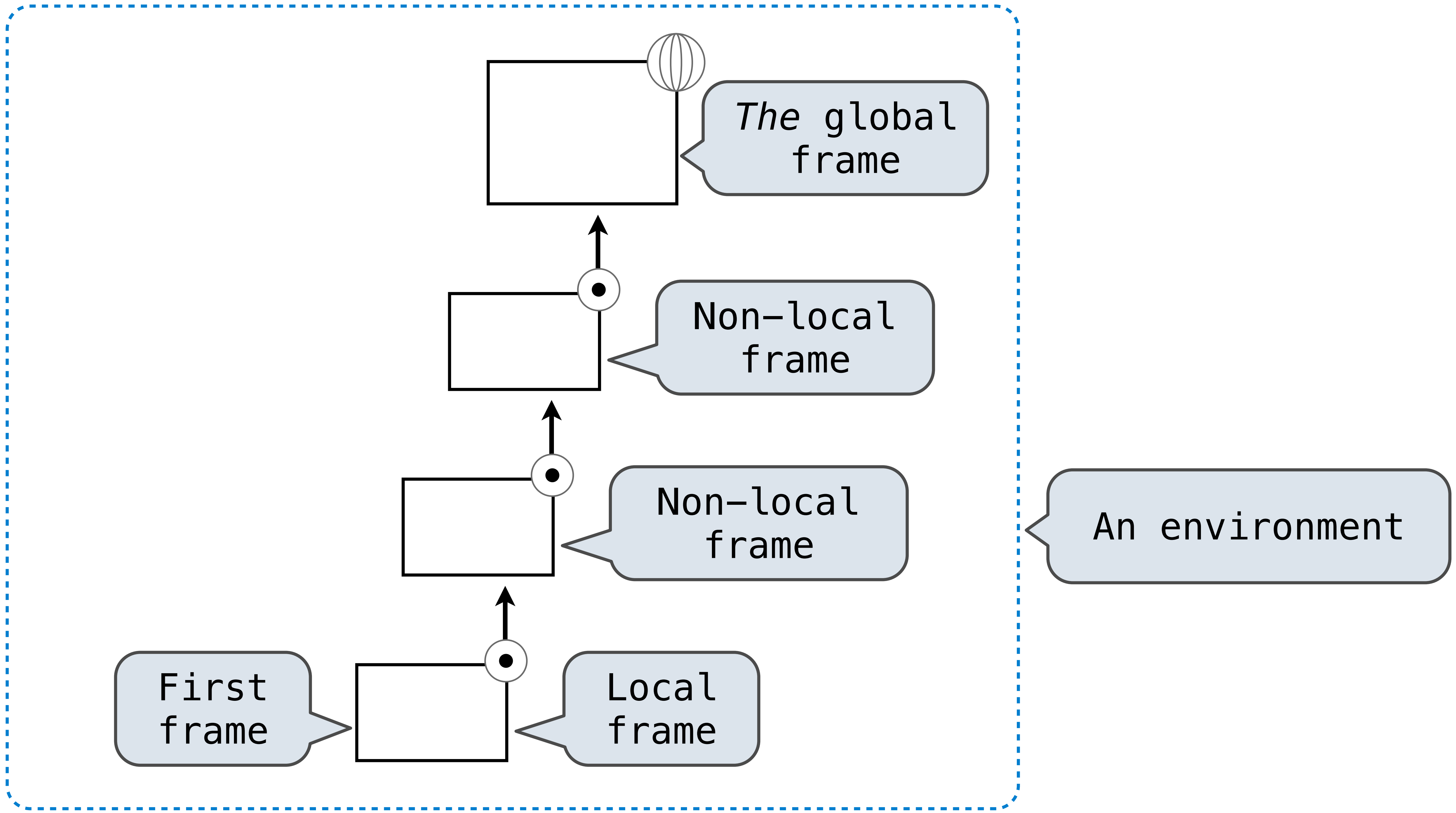
Local, Non-Local, and Global Frames



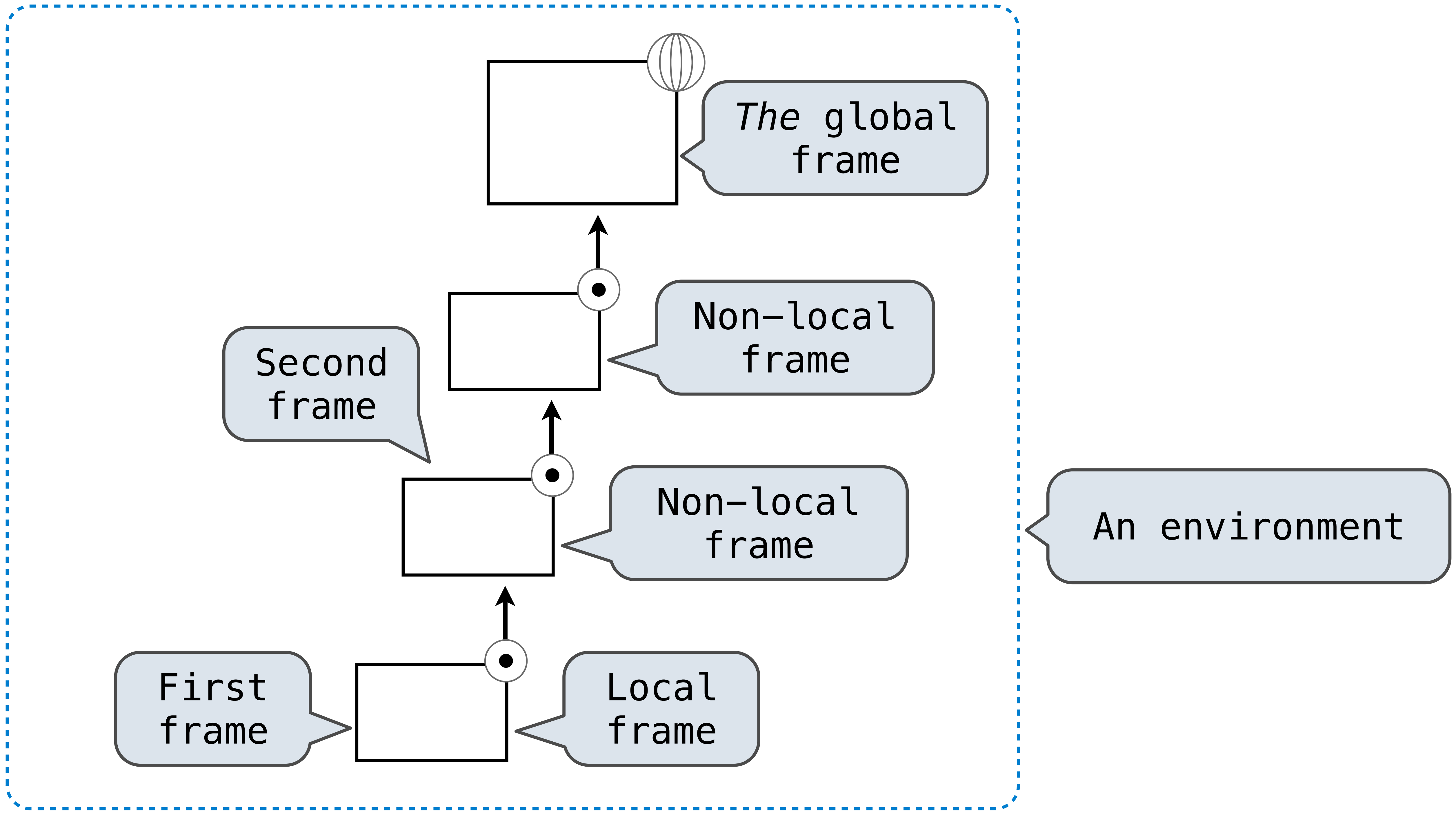
Local, Non-Local, and Global Frames



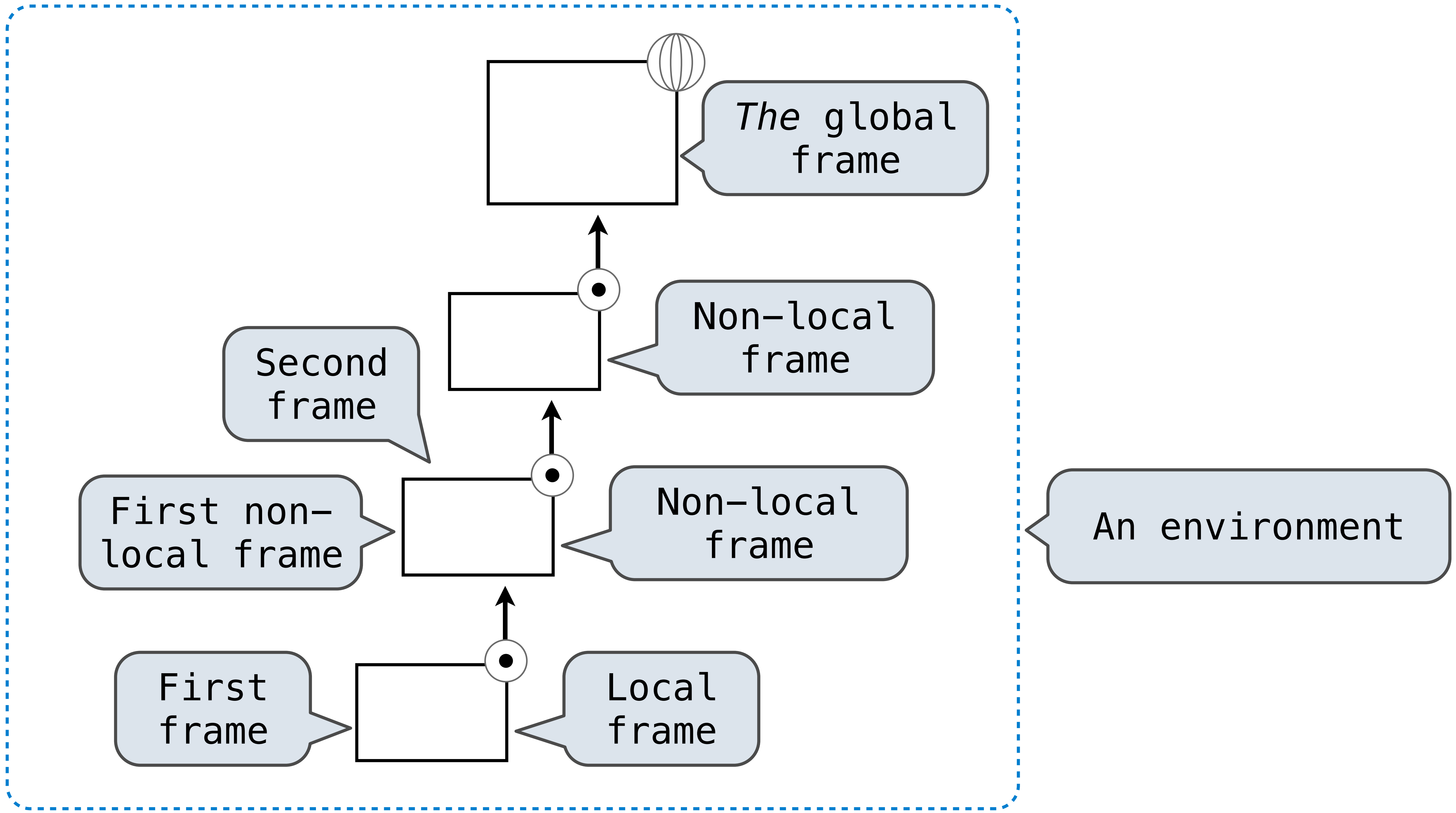
Local, Non-Local, and Global Frames



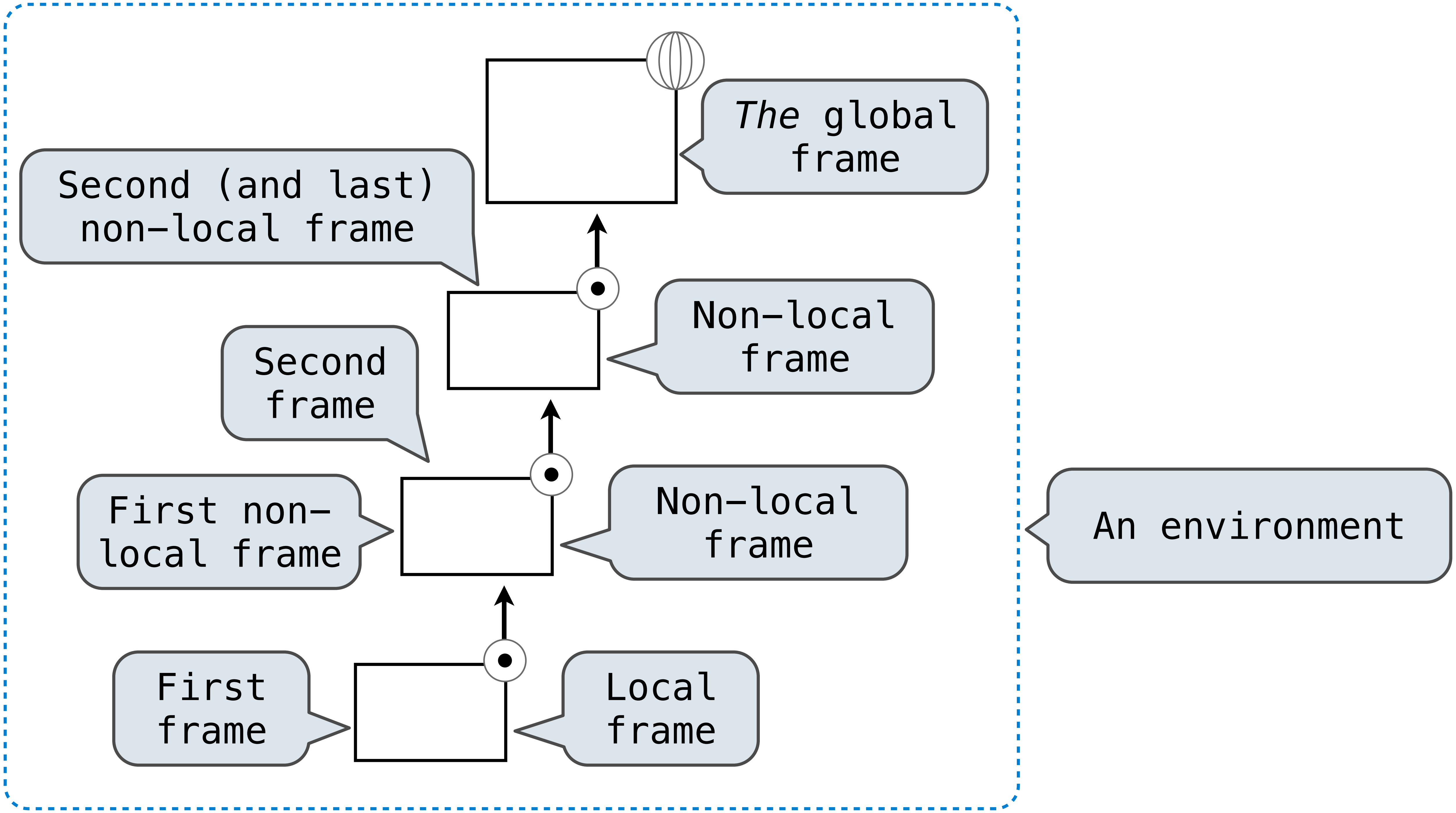
Local, Non-Local, and Global Frames



Local, Non-Local, and Global Frames



Local, Non-Local, and Global Frames



The Effect of Nonlocal Statements

```
nonlocal <name>
```

The Effect of Nonlocal Statements

```
nonlocal <name>
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

The Effect of Nonlocal Statements

```
nonlocal <name>
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

The Effect of Nonlocal Statements

```
nonlocal <name> , <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

The Effect of Nonlocal Statements

```
nonlocal <name> , <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

The Effect of Nonlocal Statements

```
nonlocal <name> , <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

The Effect of Nonlocal Statements

```
nonlocal <name> , <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

The Effect of Nonlocal Statements

```
nonlocal <name>, <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

http://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement

The Effect of Nonlocal Statements

```
nonlocal <name> , <name 2>, ...
```

Effect: Future references to that name refer to its pre-existing binding in the **first non-local frame** of the current environment in which that name is bound.

Python Docs: an "enclosing scope"

From the Python 3 language reference:

Names listed in a nonlocal statement must refer to pre-existing bindings in an enclosing scope.

Names listed in a nonlocal statement must not collide with pre-existing bindings in the local scope.

http://docs.python.org/release/3.1.3/reference/simple_stmts.html#the-nonlocal-statement

http://www.python.org/dev/peps/pep-3104/

The Many Meanings of Assignment Statements

```
x = 2
```

The Many Meanings of Assignment Statements

$\boxed{\texttt{x = 2}}$

Status	Effect

The Many Meanings of Assignment Statements

```
x = 2
```

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	

The Many Meanings of Assignment Statements

```
x = 2
```

Status	Effect
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.

The Many Meanings of Assignment Statements

```
x = 2
```

Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	

The Many Meanings of Assignment Statements

`x = 2`

Status	Effect
•No nonlocal statement •"x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
•No nonlocal statement •"x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.

The Many Meanings of Assignment Statements

```
x = 2
```

Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	

The Many Meanings of Assignment Statements

```
x = 2
```

Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound.

The Many Meanings of Assignment Statements

```
x = 2
```

Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound.
• nonlocal x • "x" **is not** bound in a non-local frame	

The Many Meanings of Assignment Statements

```
x = 2
```

Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound.
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found

The Many Meanings of Assignment Statements

```
x = 2
```

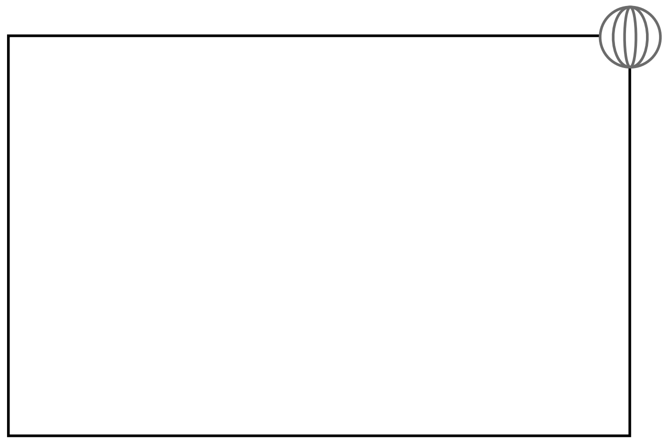
Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound.
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found
• nonlocal x • "x" **is** bound in a non-local frame • "x" also bound locally	

The Many Meanings of Assignment Statements

```
x = 2
```

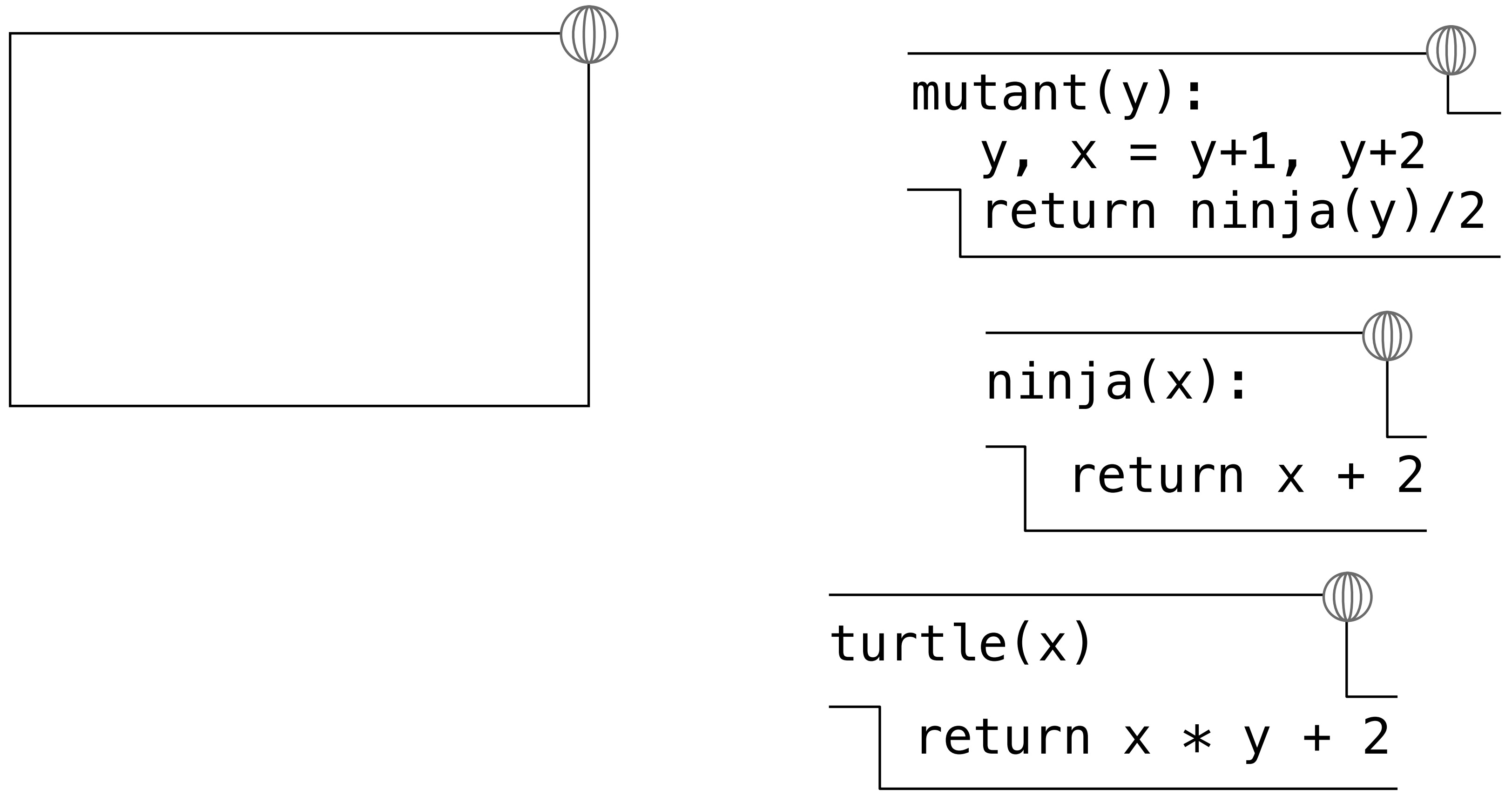
Status	**Effect**
• No nonlocal statement • "x" **is not** bound locally	Create a new binding from name "x" to object 2 in the first frame of the current environment.
• No nonlocal statement • "x" **is** bound locally	Re-bind name "x" to object 2 in the first frame of the current env.
• nonlocal x • "x" **is** bound in a non-local frame	Re-bind "x" to 2 in the first non-local frame of the current environment in it is bound.
• nonlocal x • "x" **is not** bound in a non-local frame	SyntaxError: no binding for nonlocal 'x' found
• nonlocal x • "x" **is** bound in a non-local frame • "x" also bound locally	SyntaxError: name 'x' is parameter and nonlocal

Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



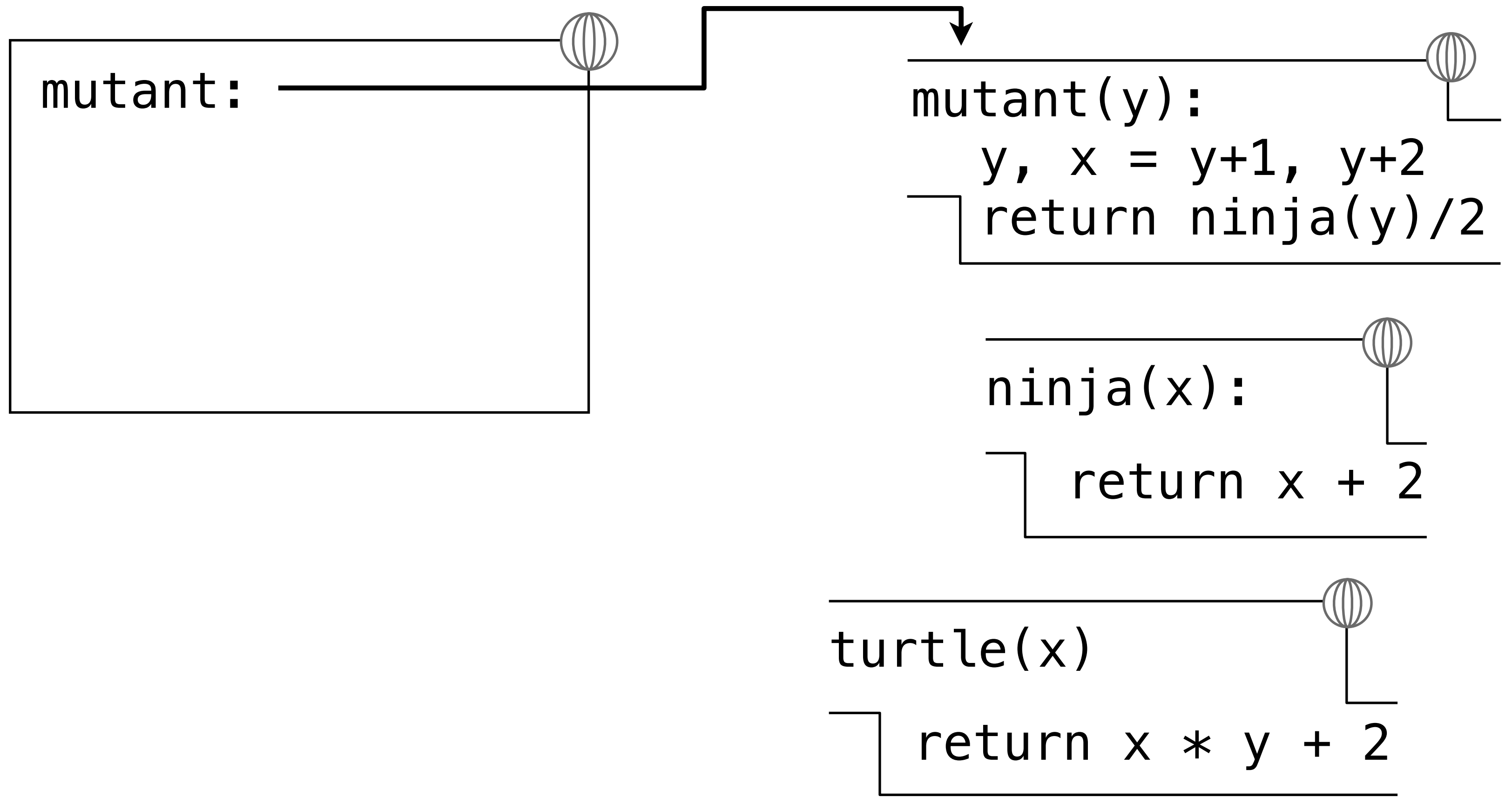
```
mutant(y):
   y, x = y+1, y+2
   return ninja(y)/2
```

```
ninja(x):

   return x + 2
```

```
turtle(x)

   return x * y + 2
```

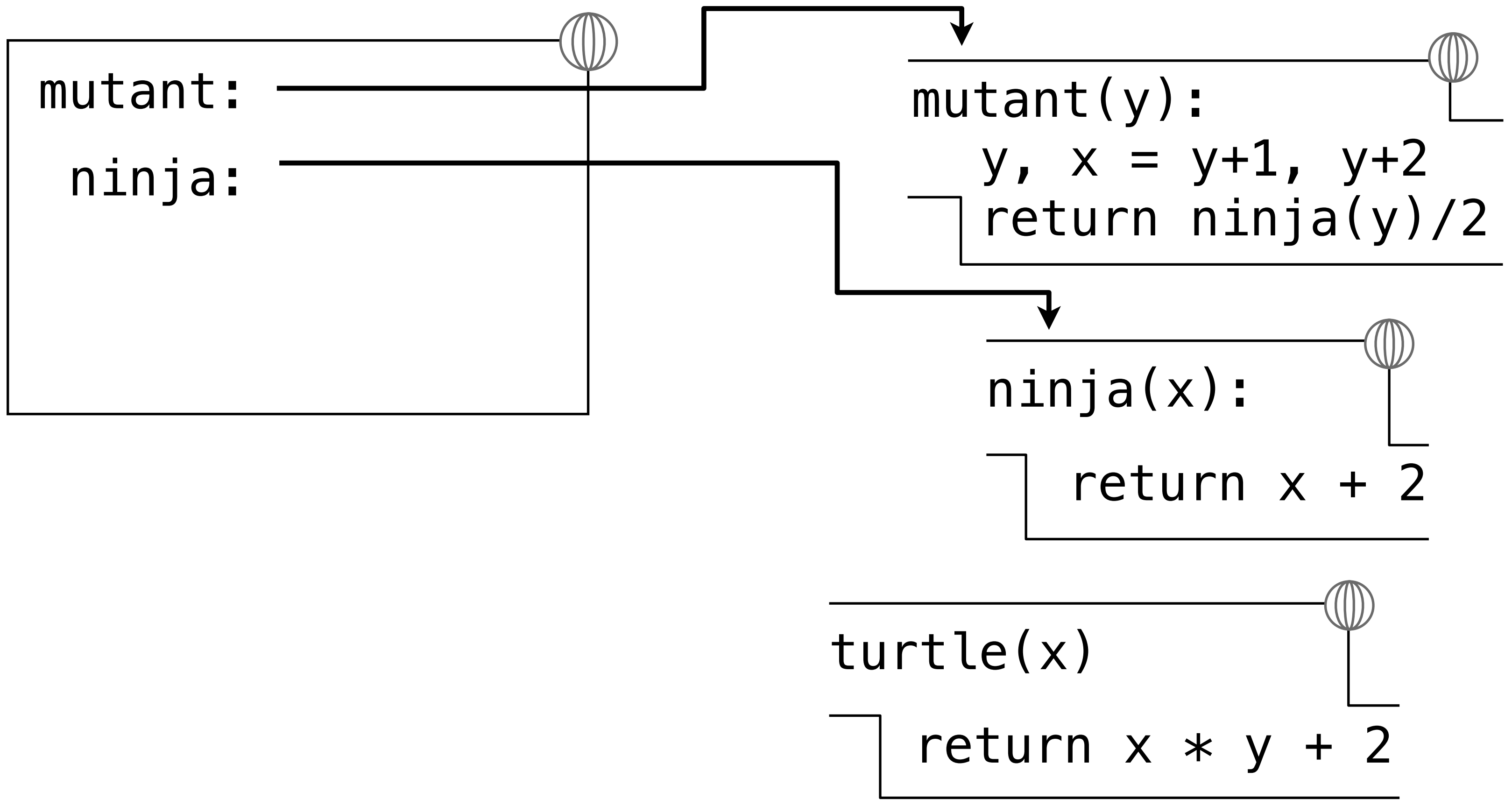
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



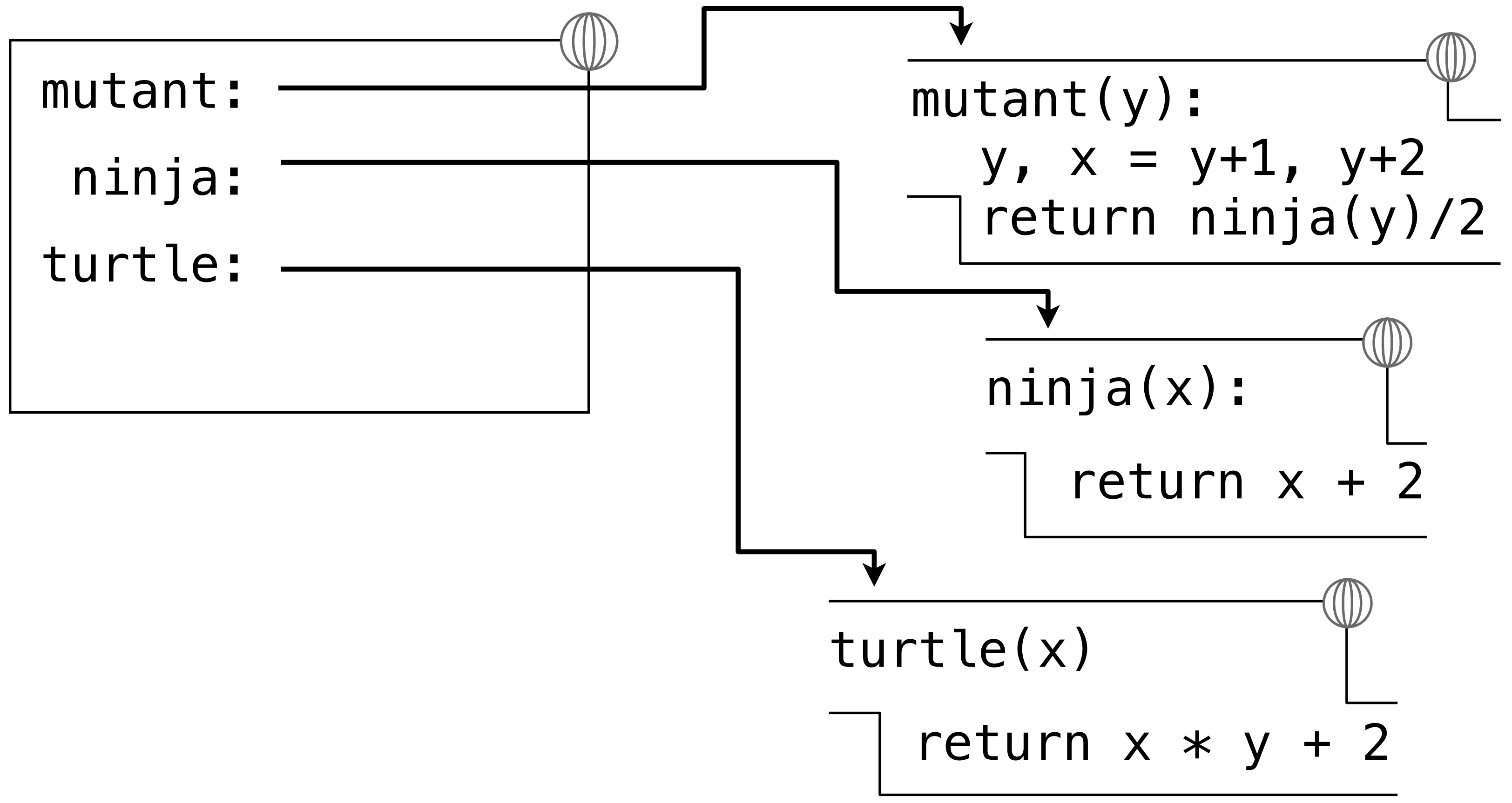
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



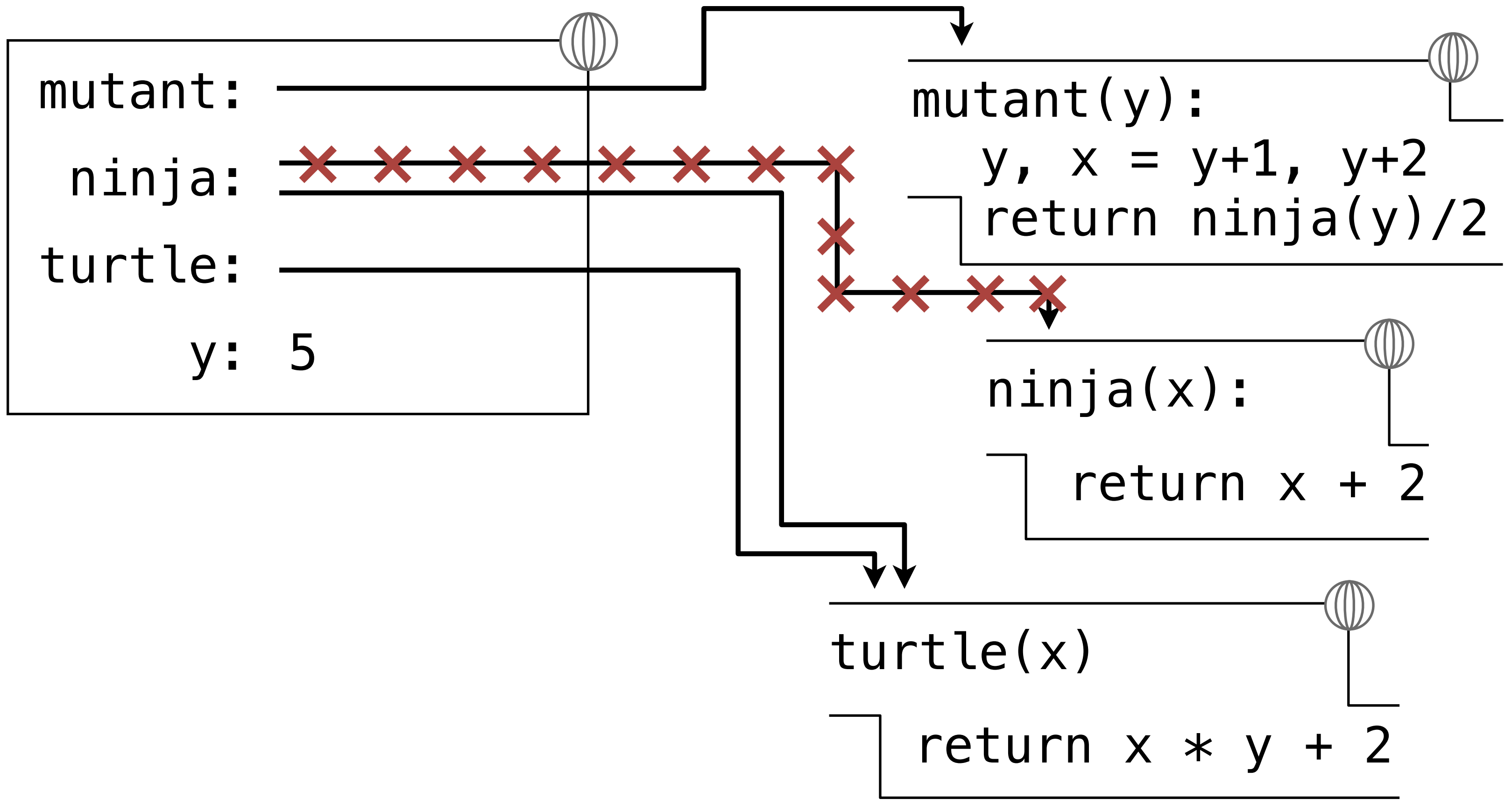
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



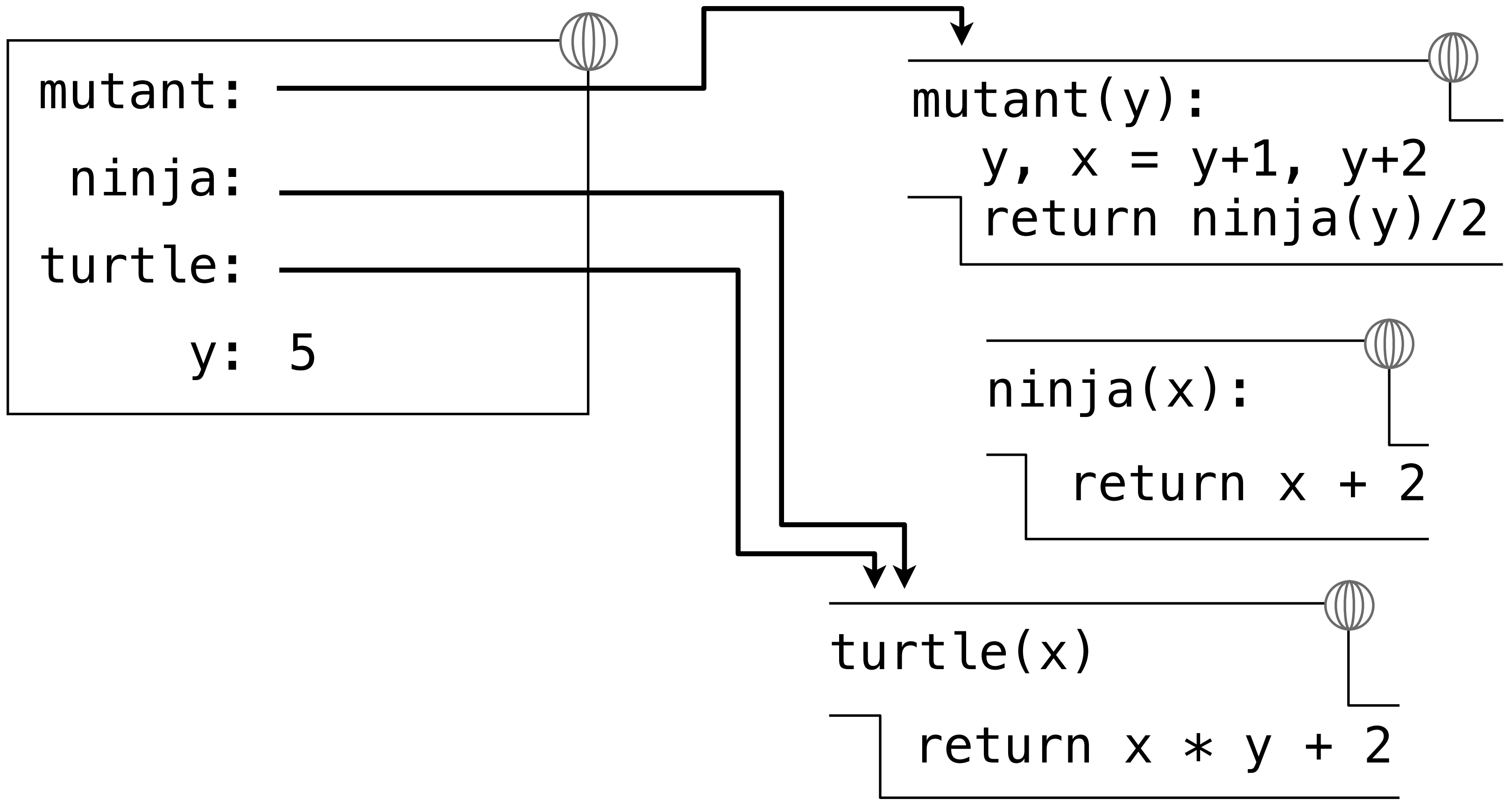
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



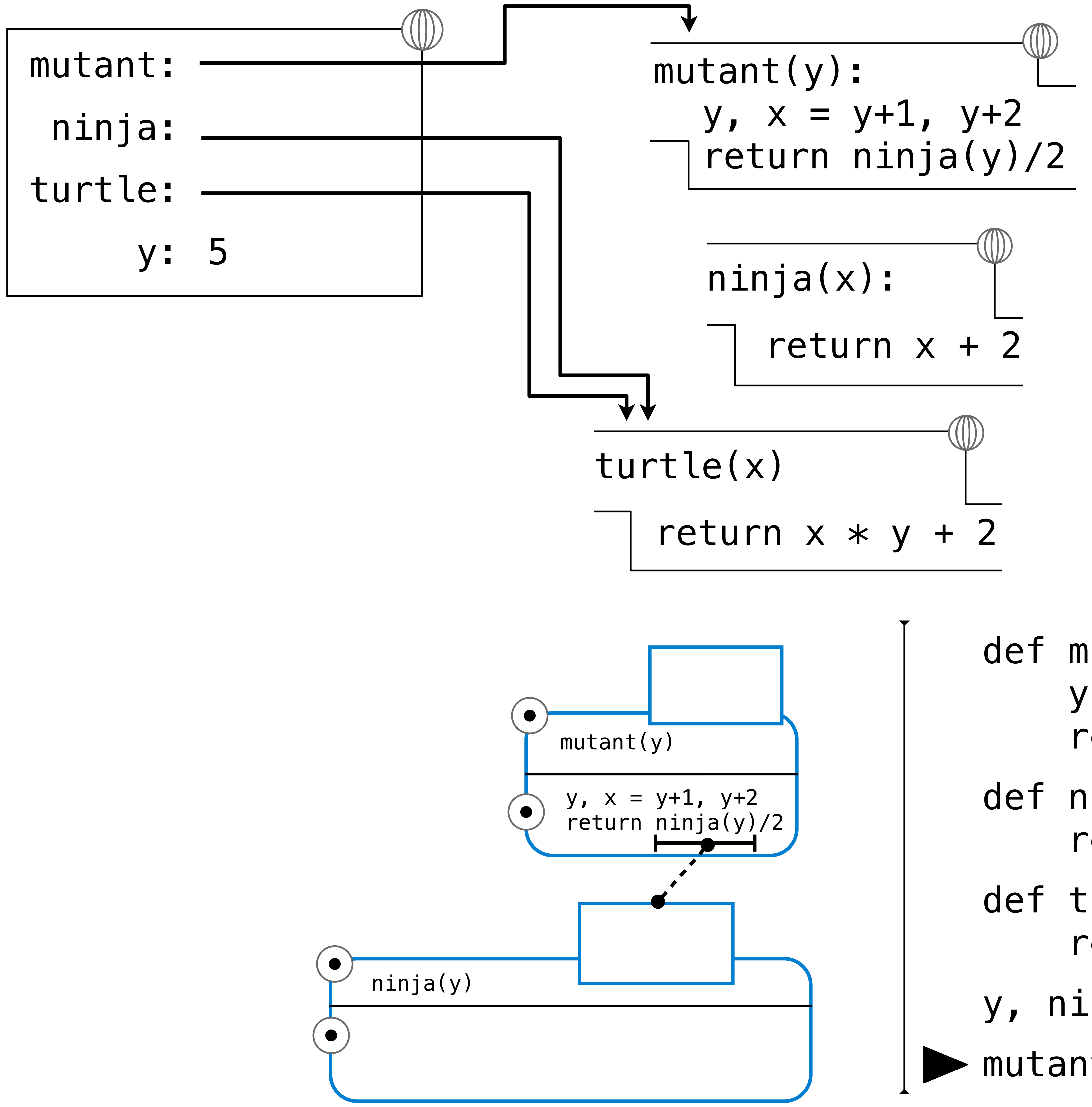
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



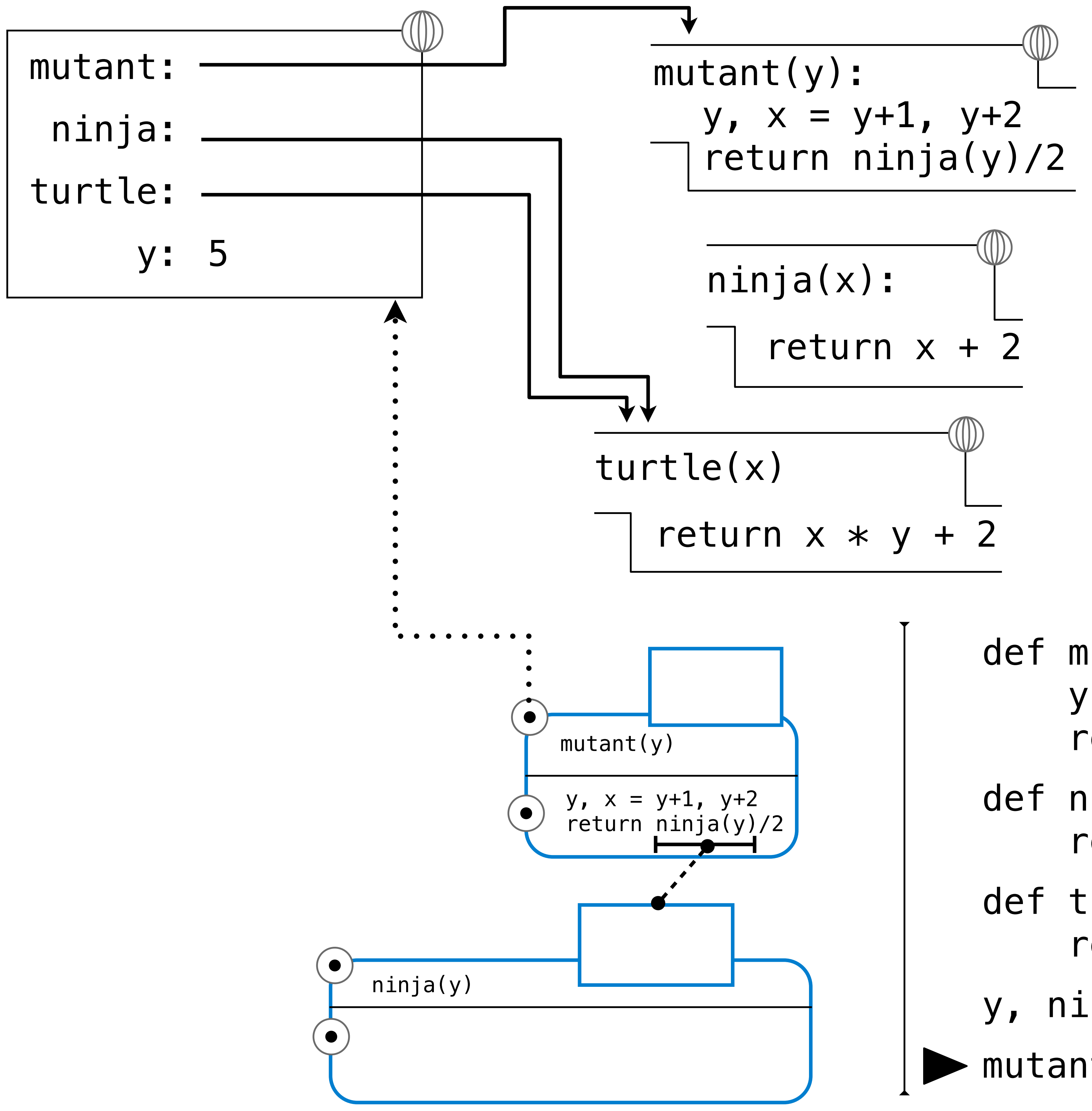
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



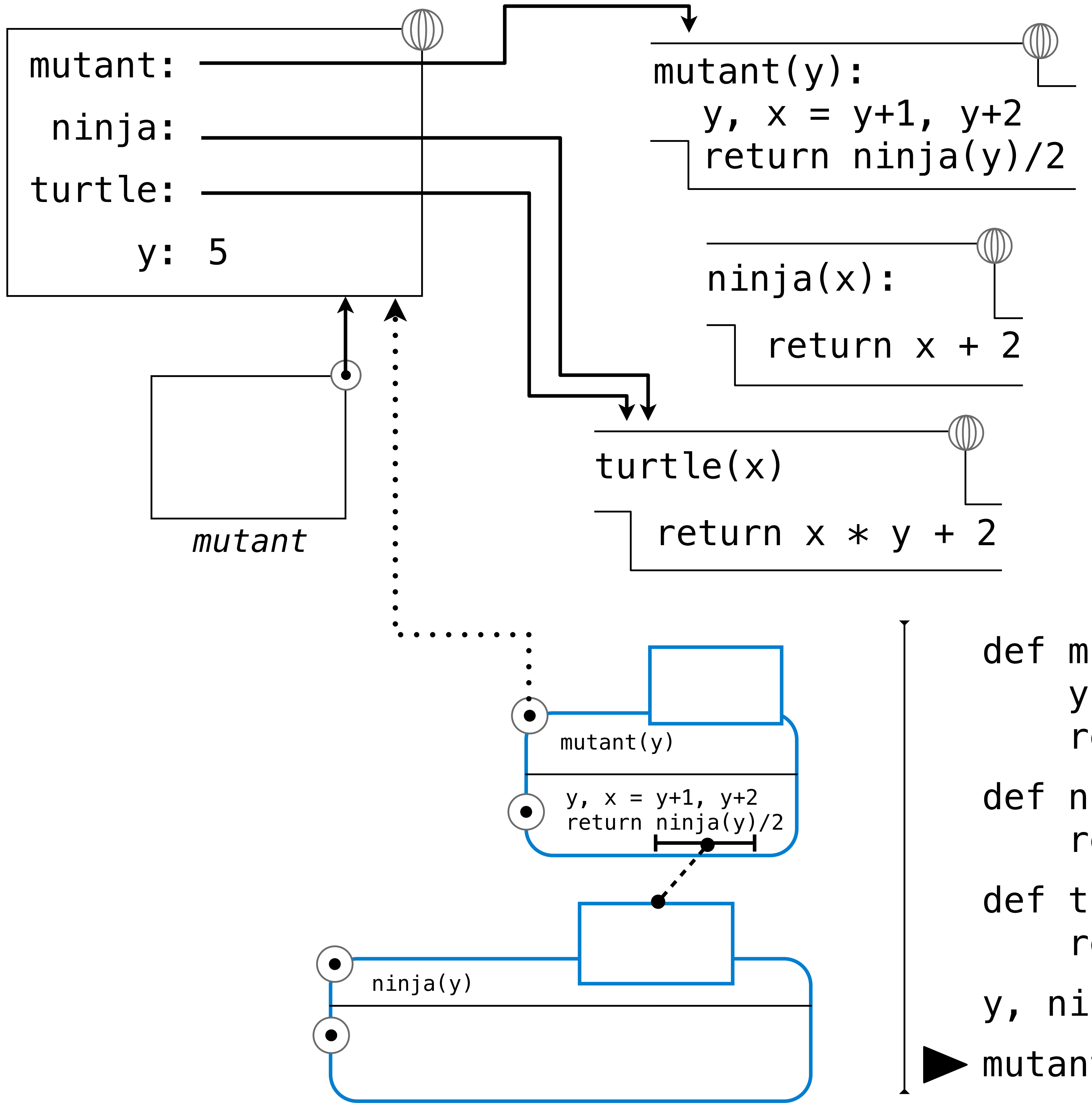
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



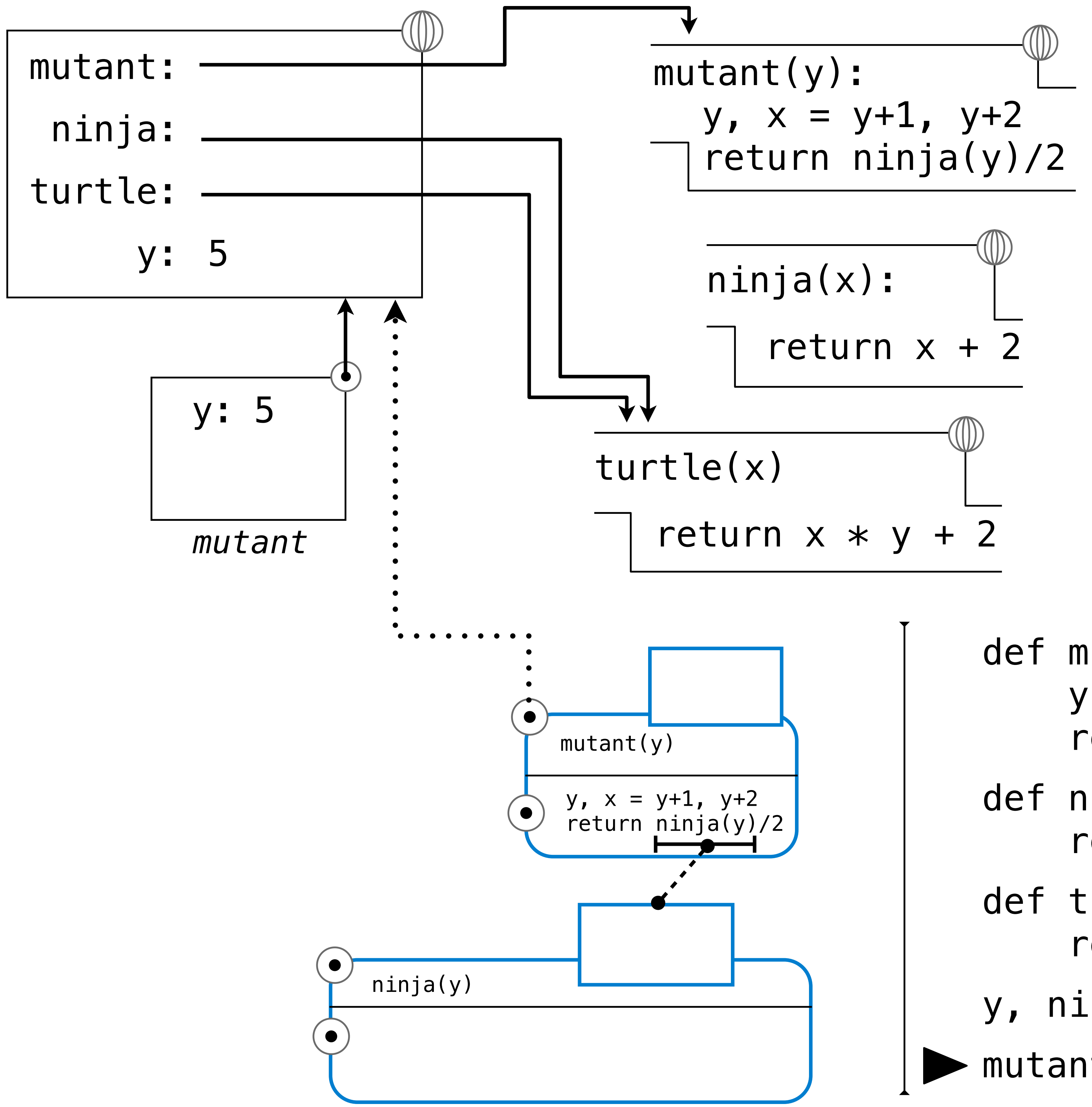
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



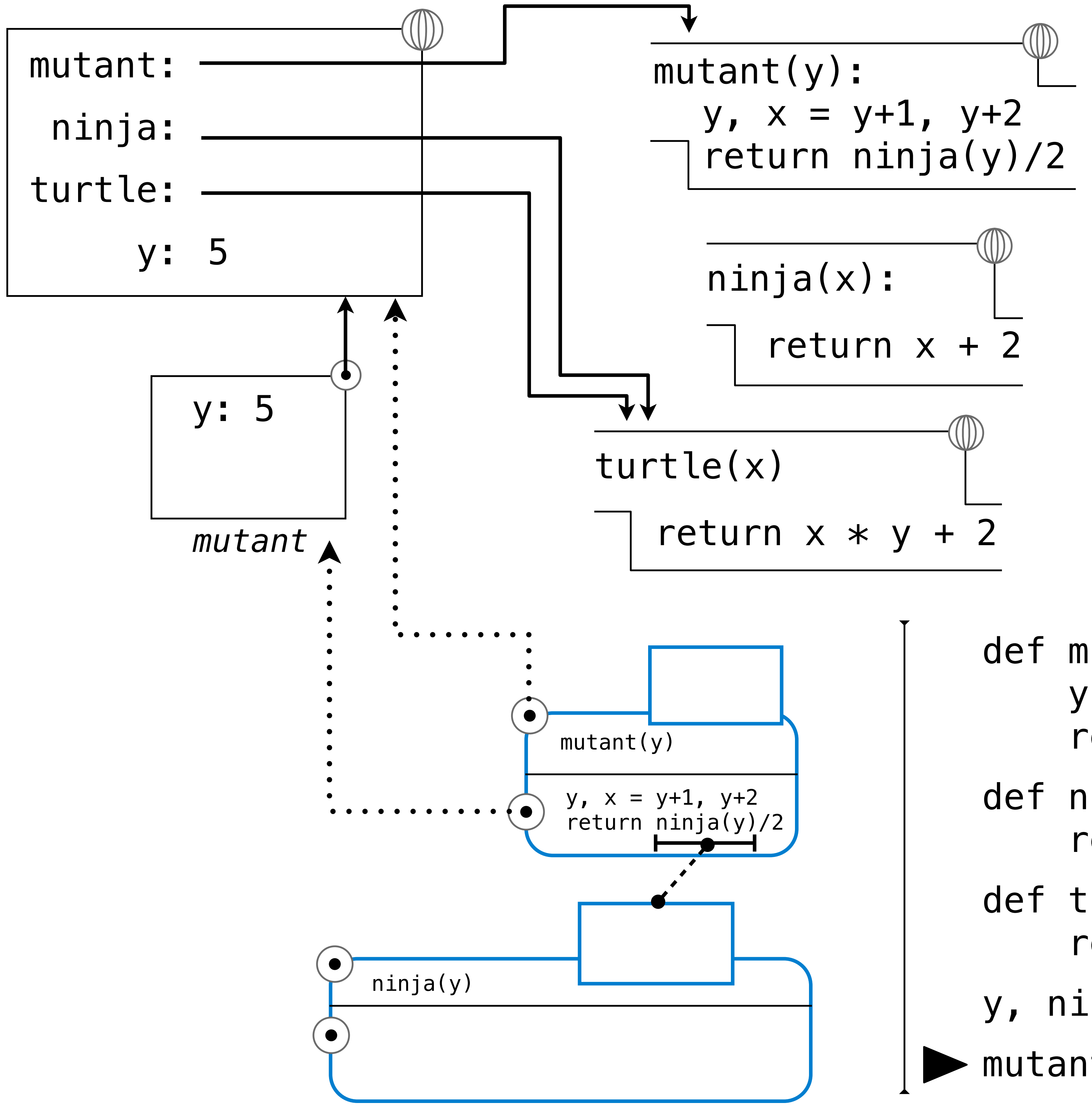
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



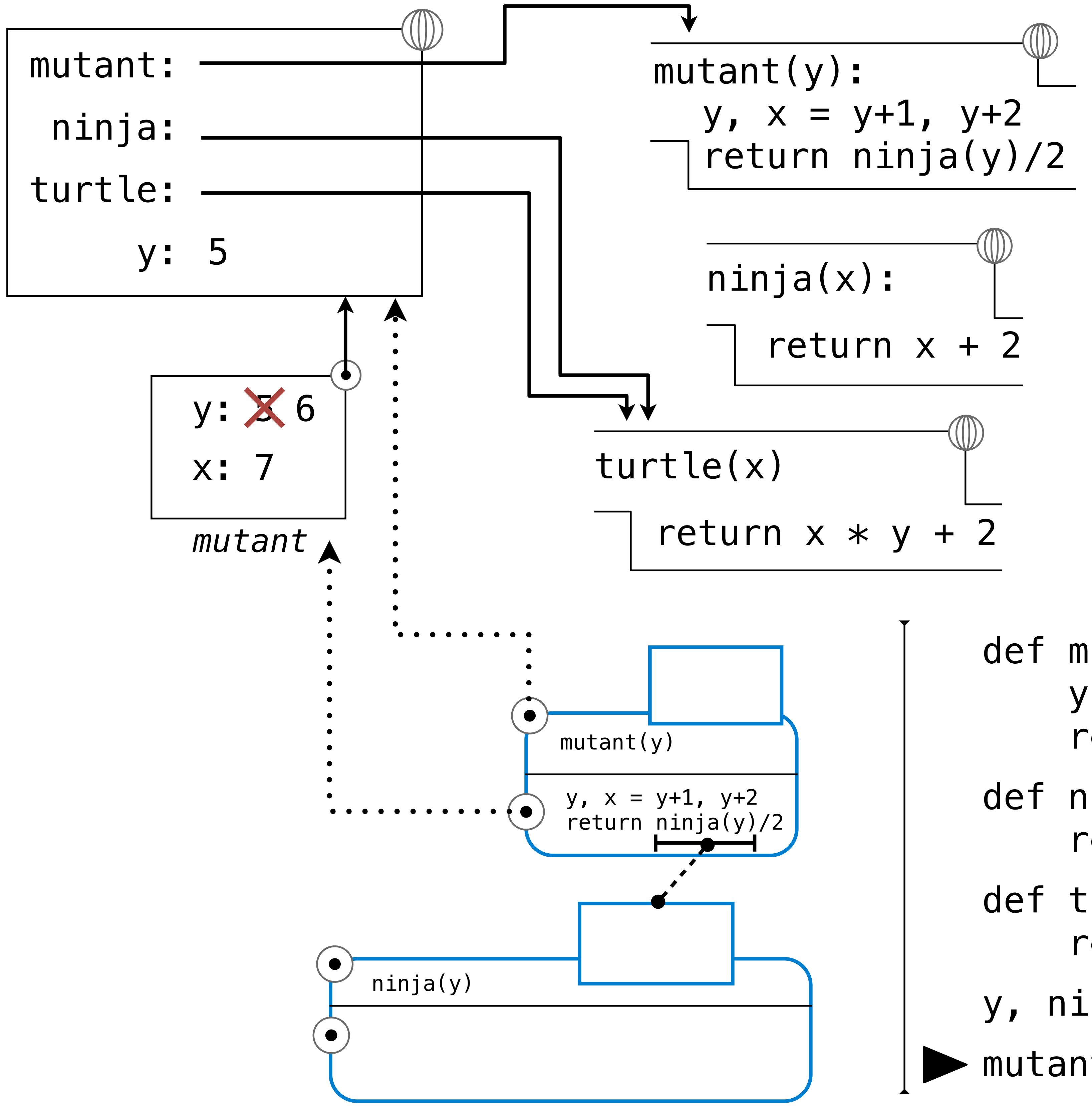
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



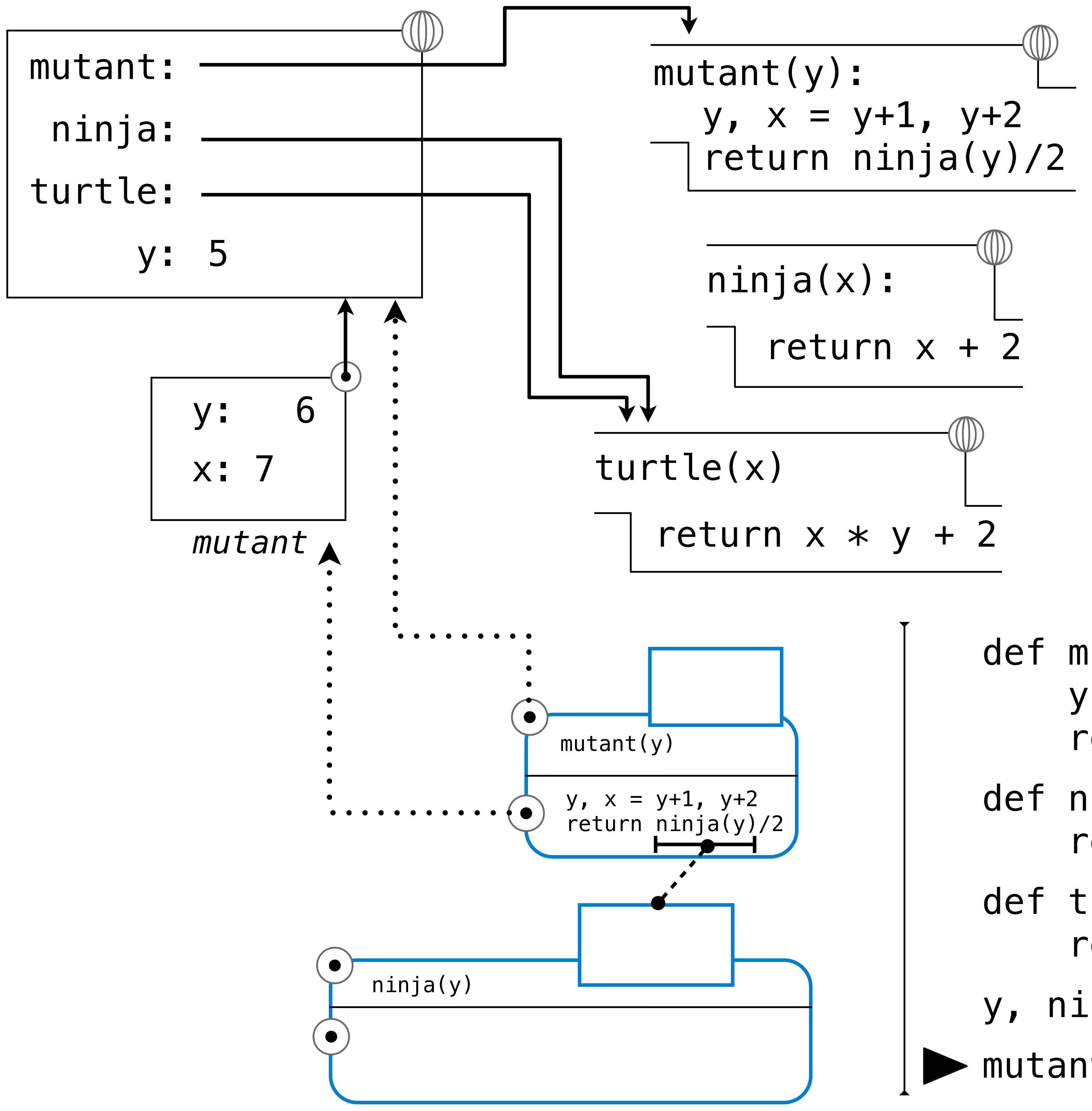
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



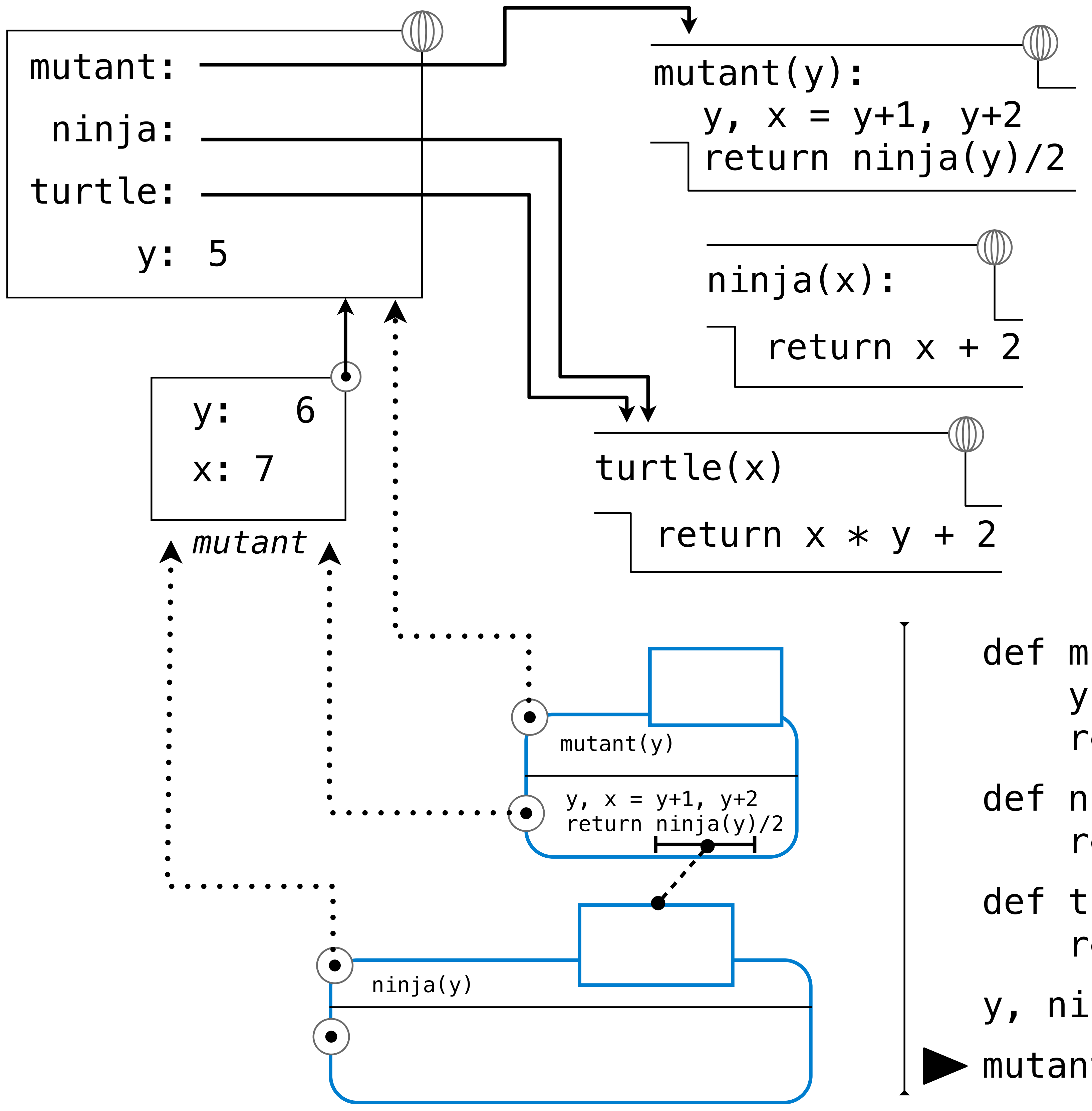
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

▶ mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



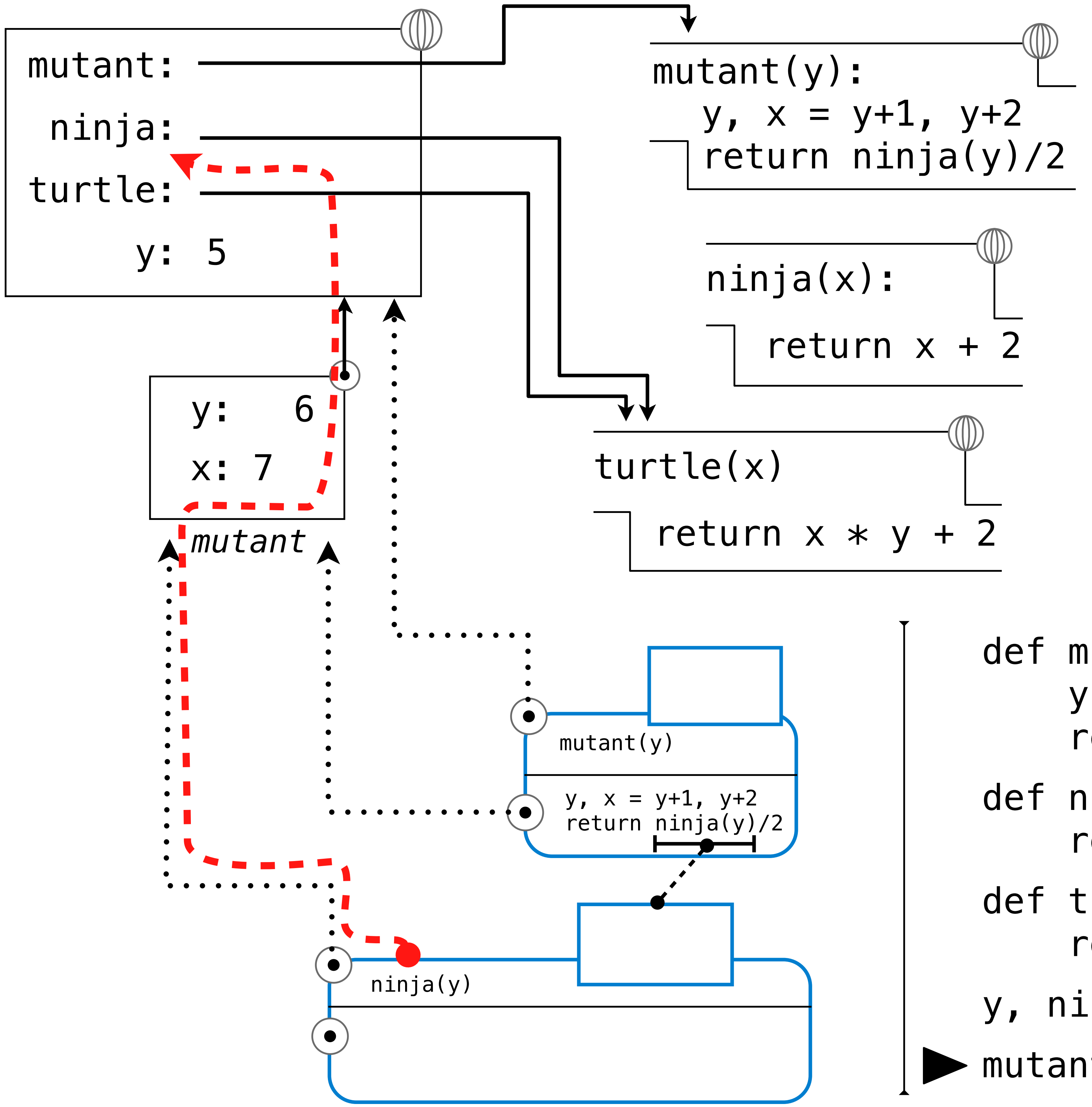
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



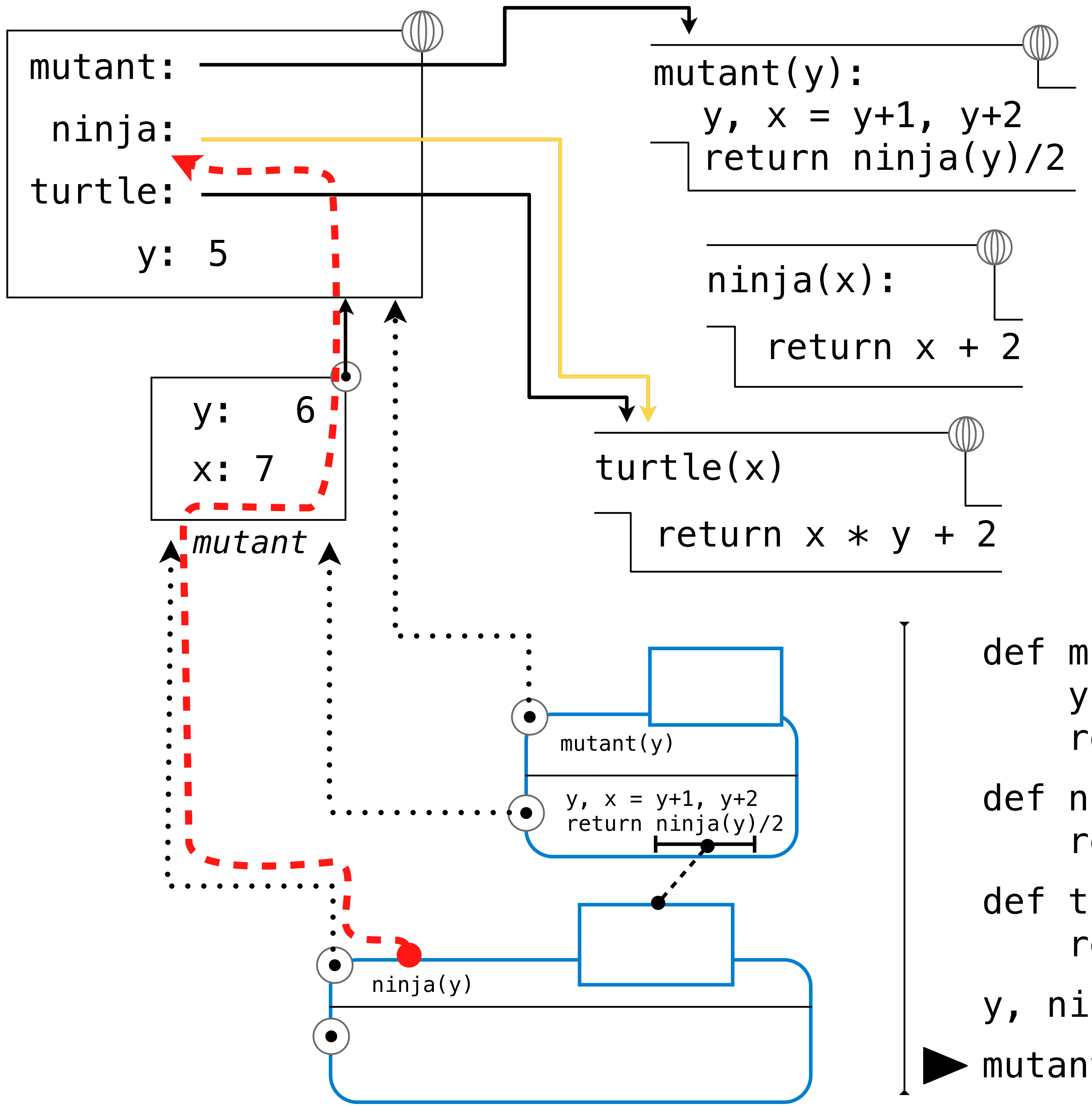
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



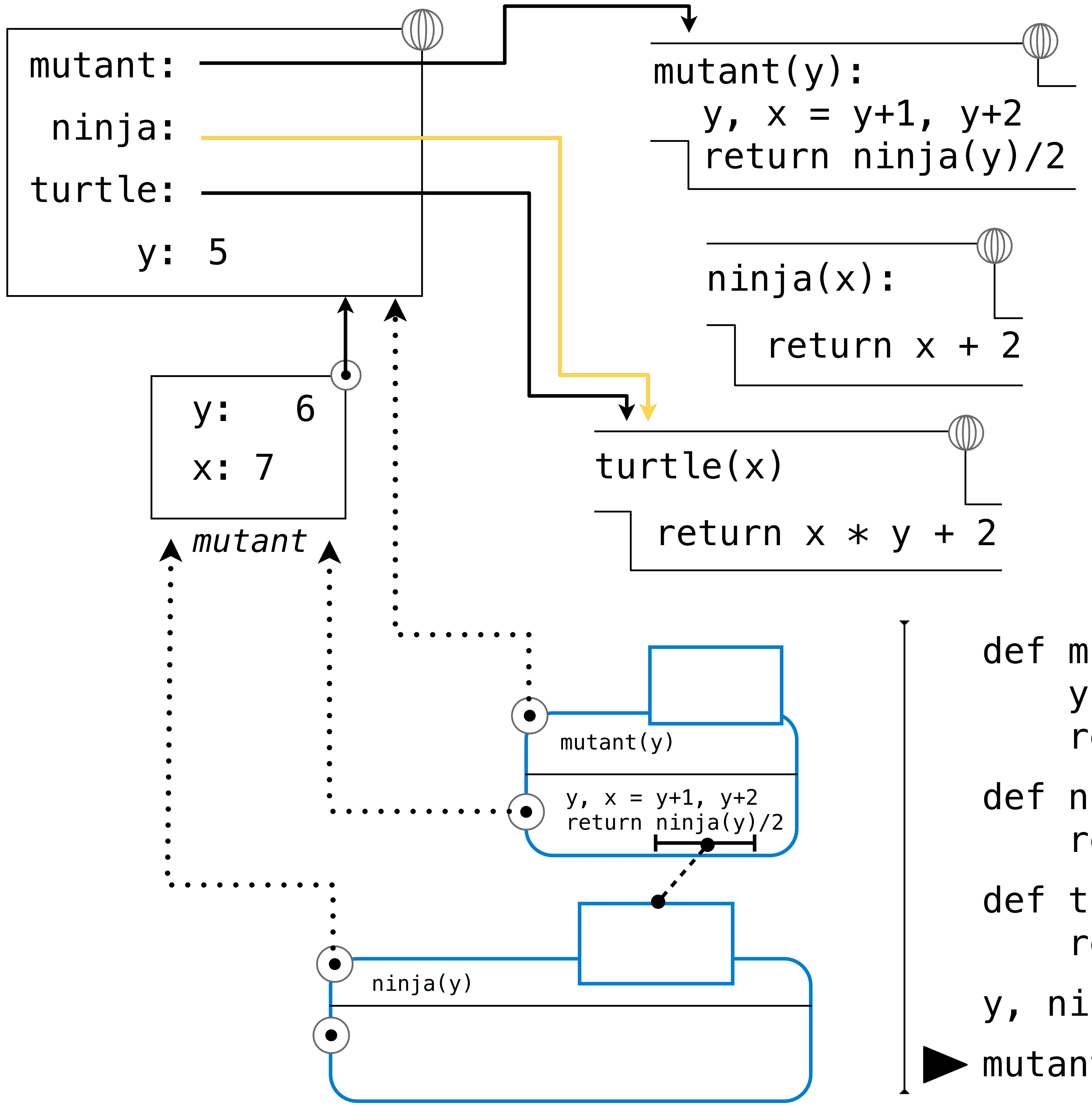
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

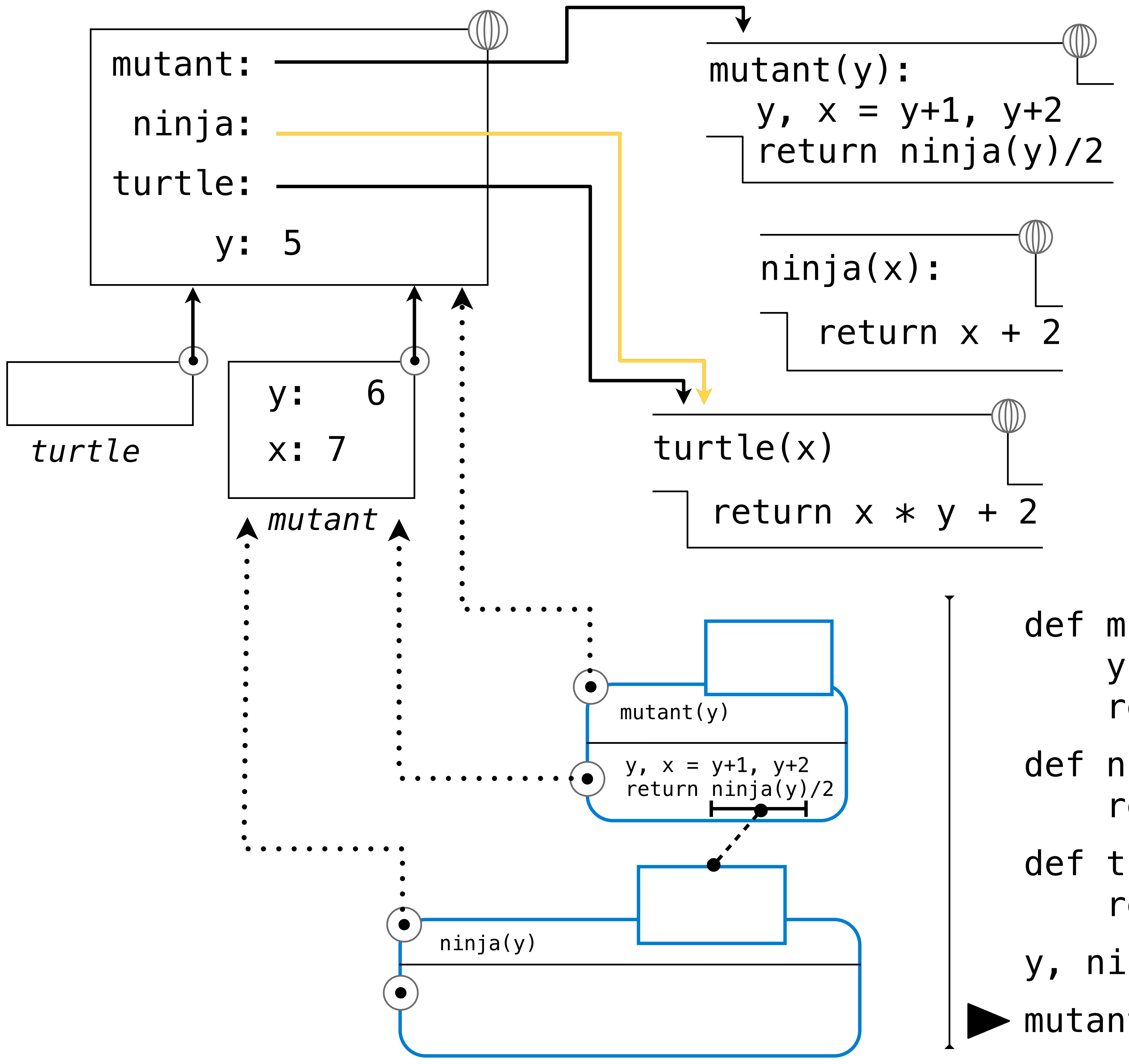
Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

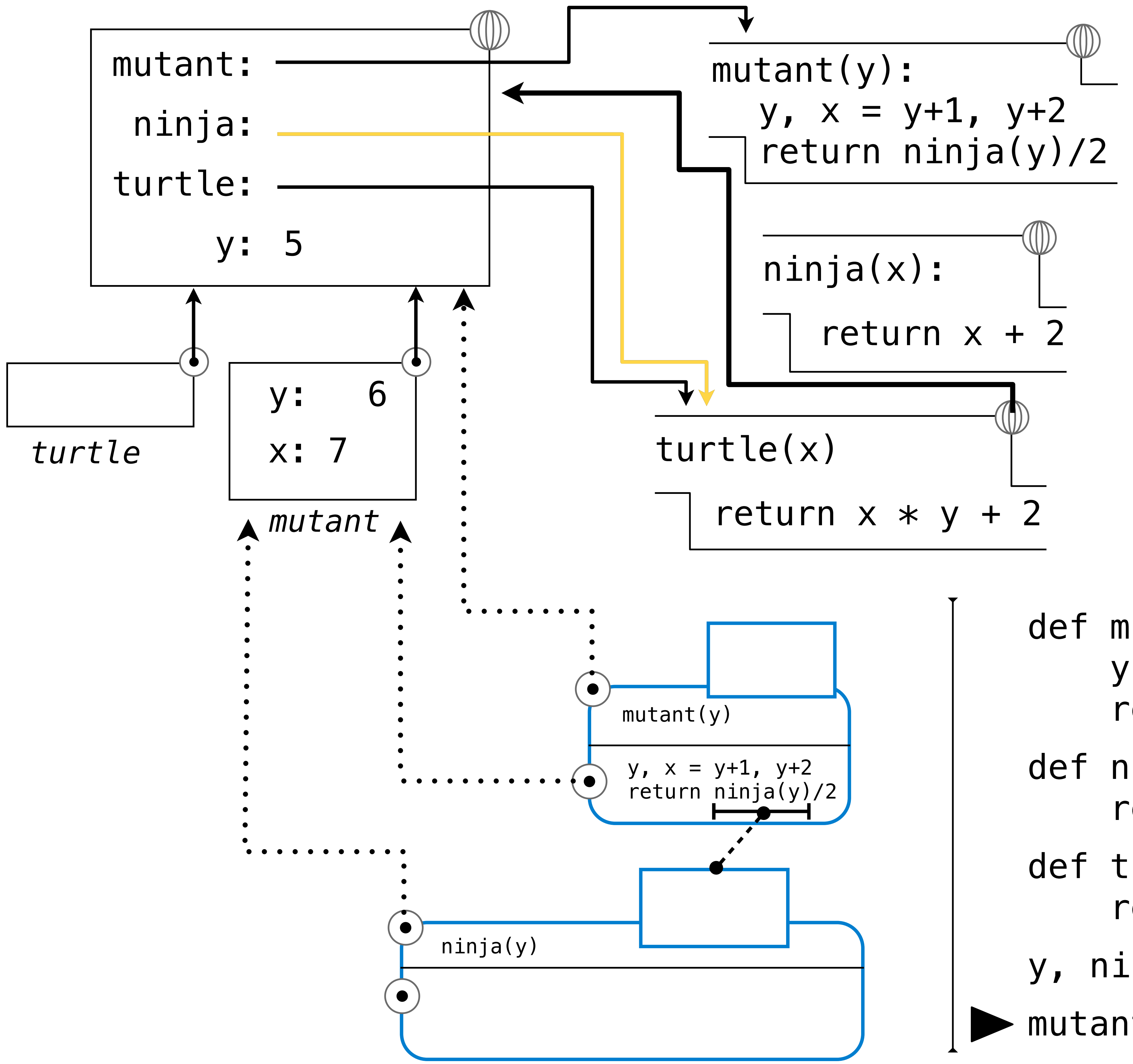
Friday, September 23, 2011

Assignment Review: Teenage Mutant Ninja Turtles



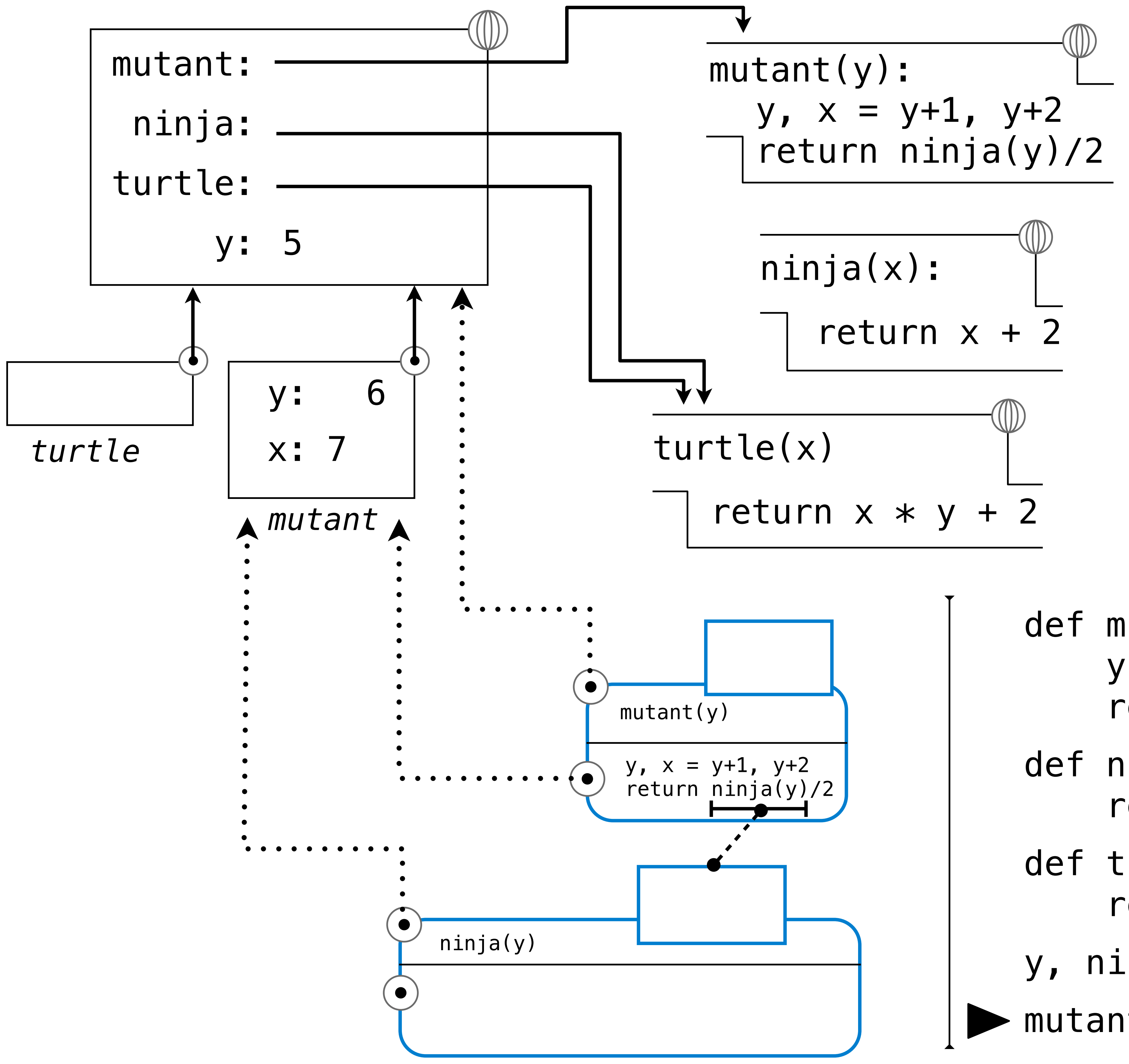
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

▶ mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



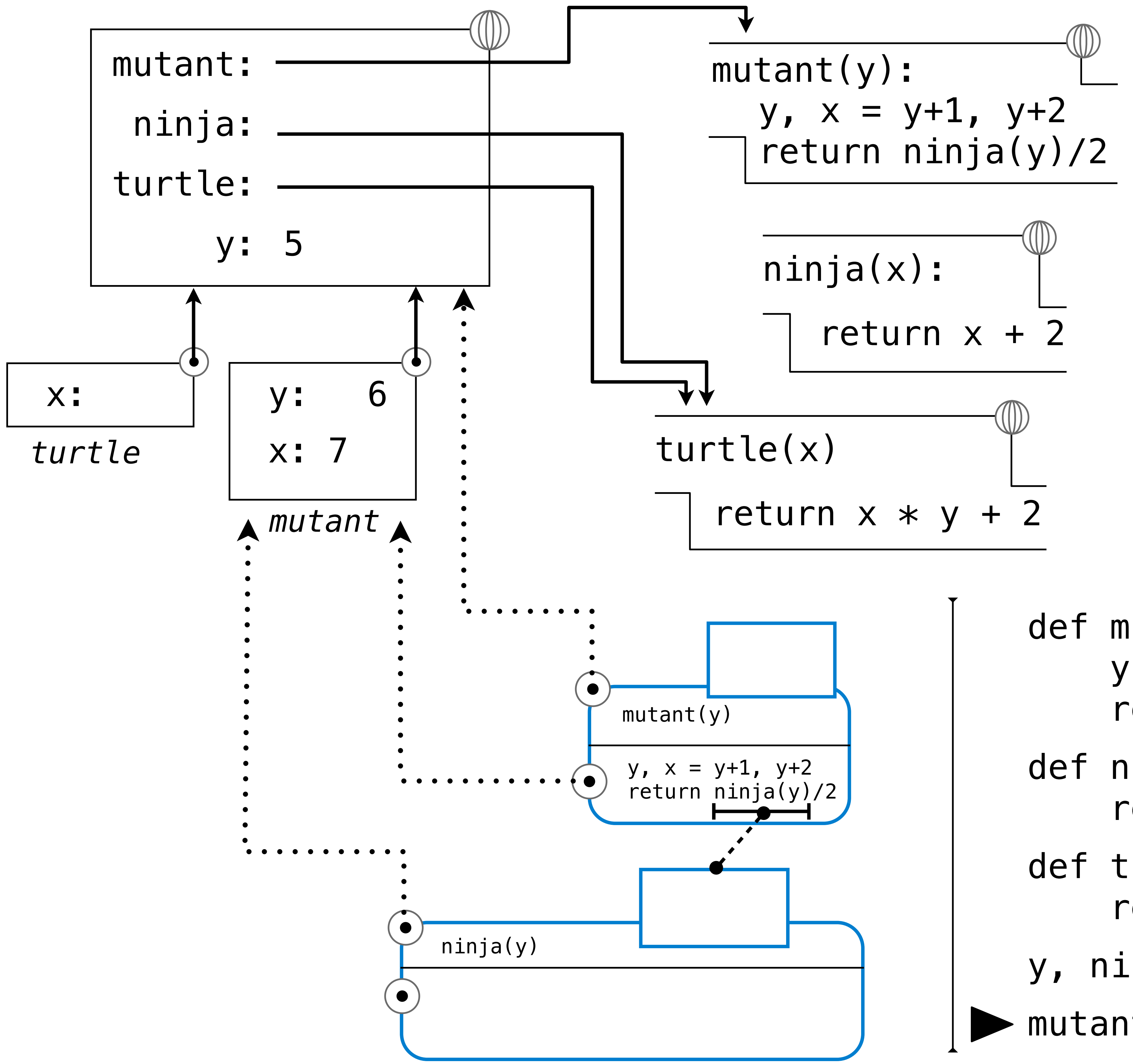
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



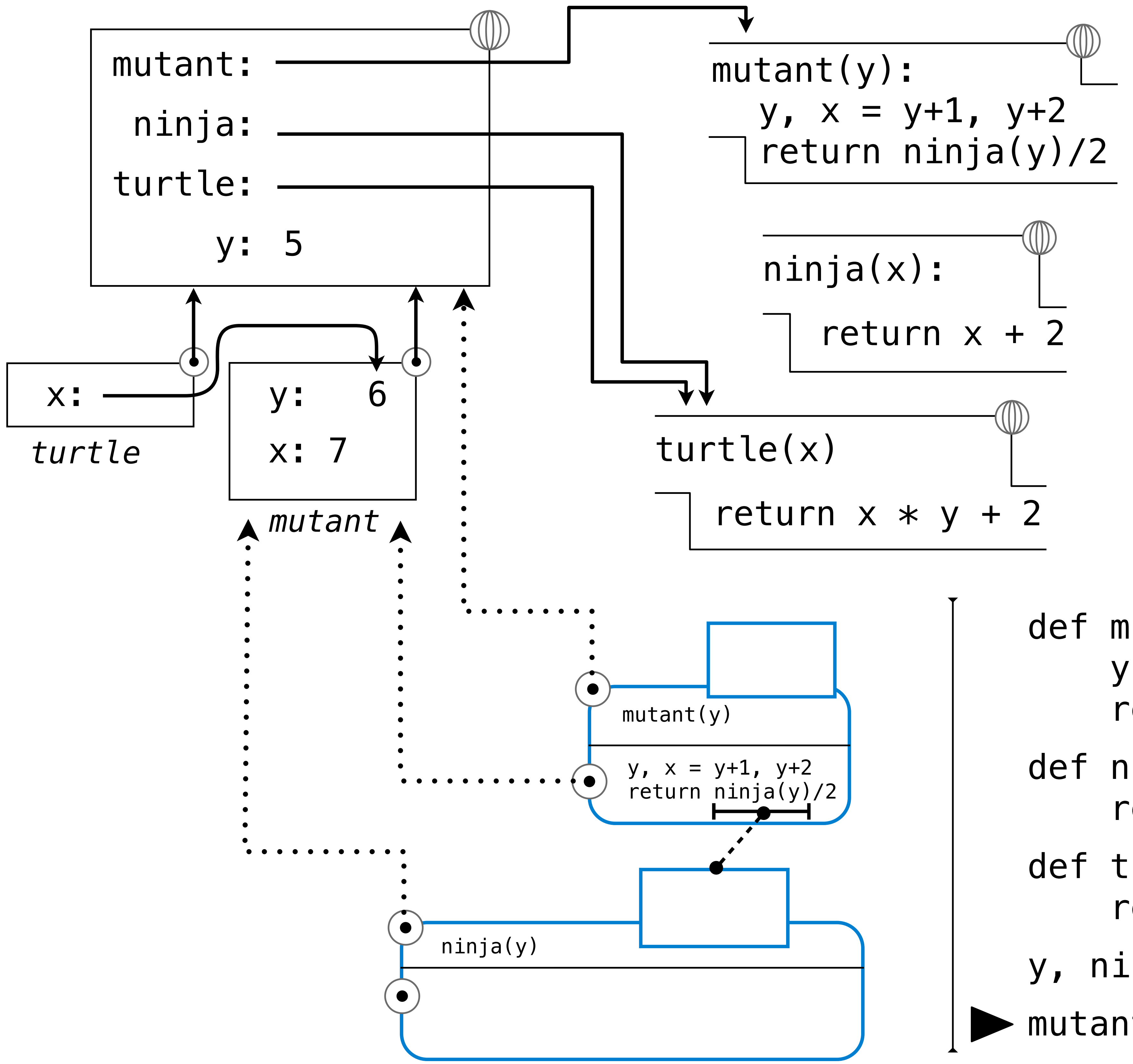
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

▶ mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



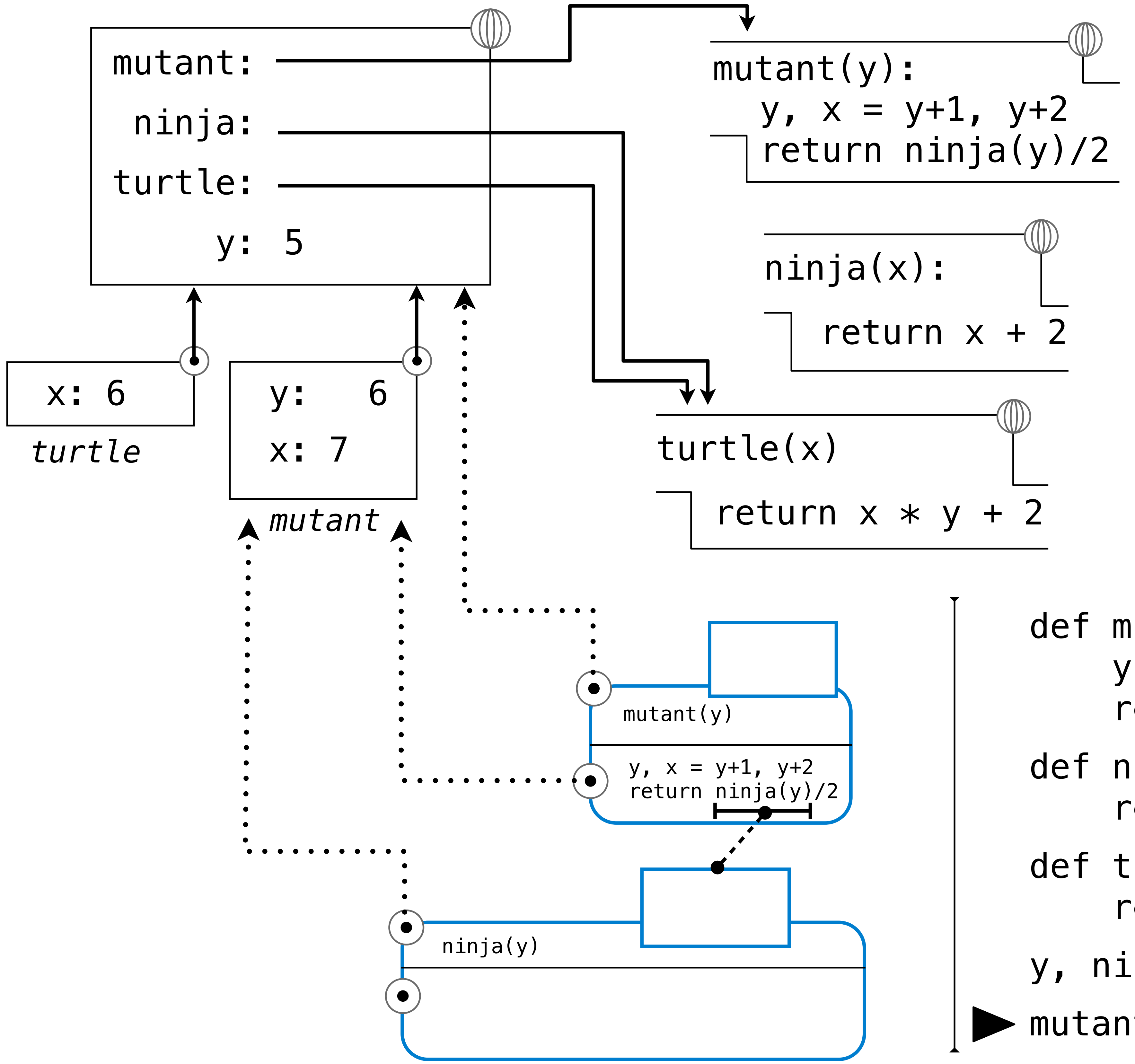
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



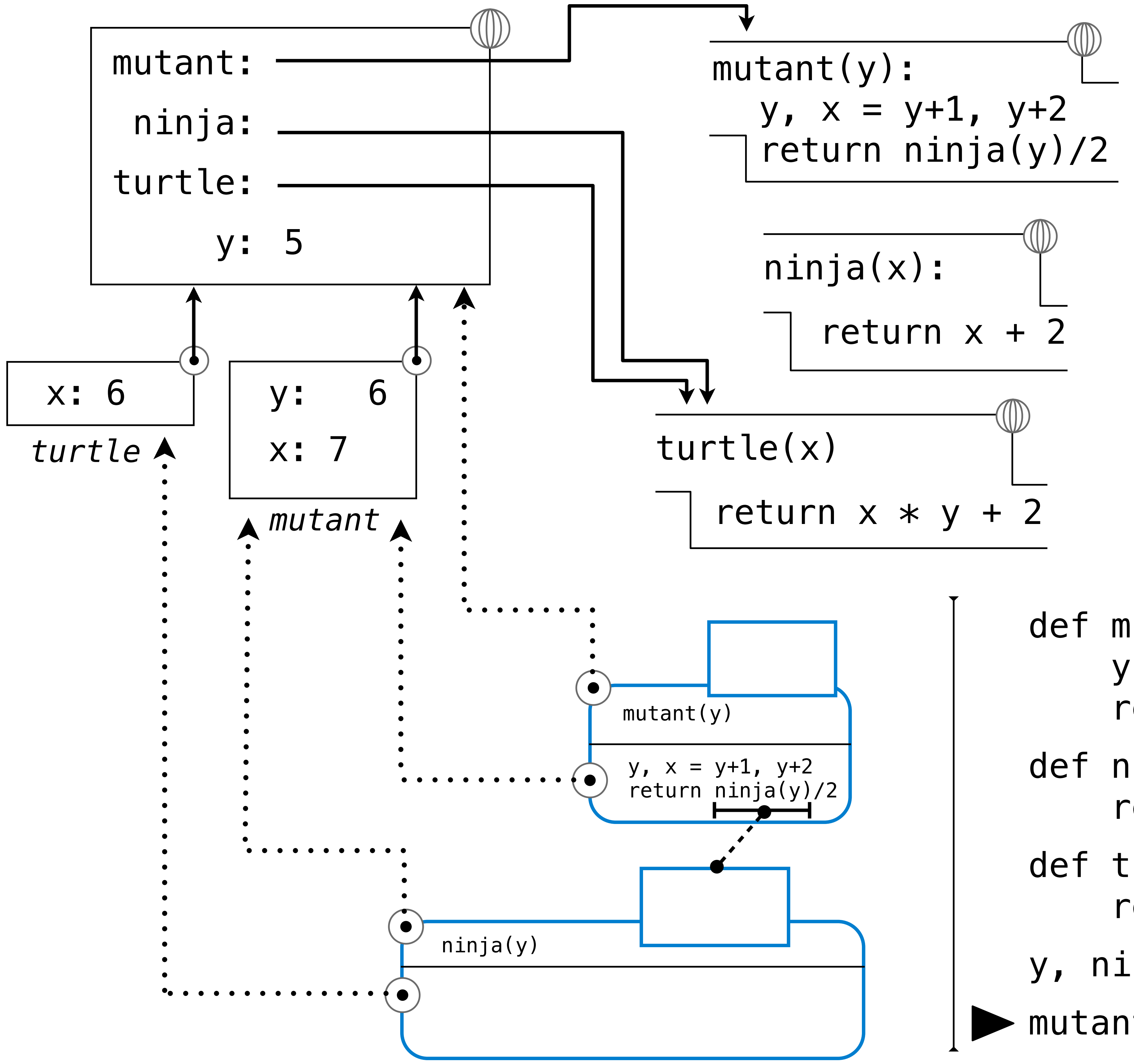
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



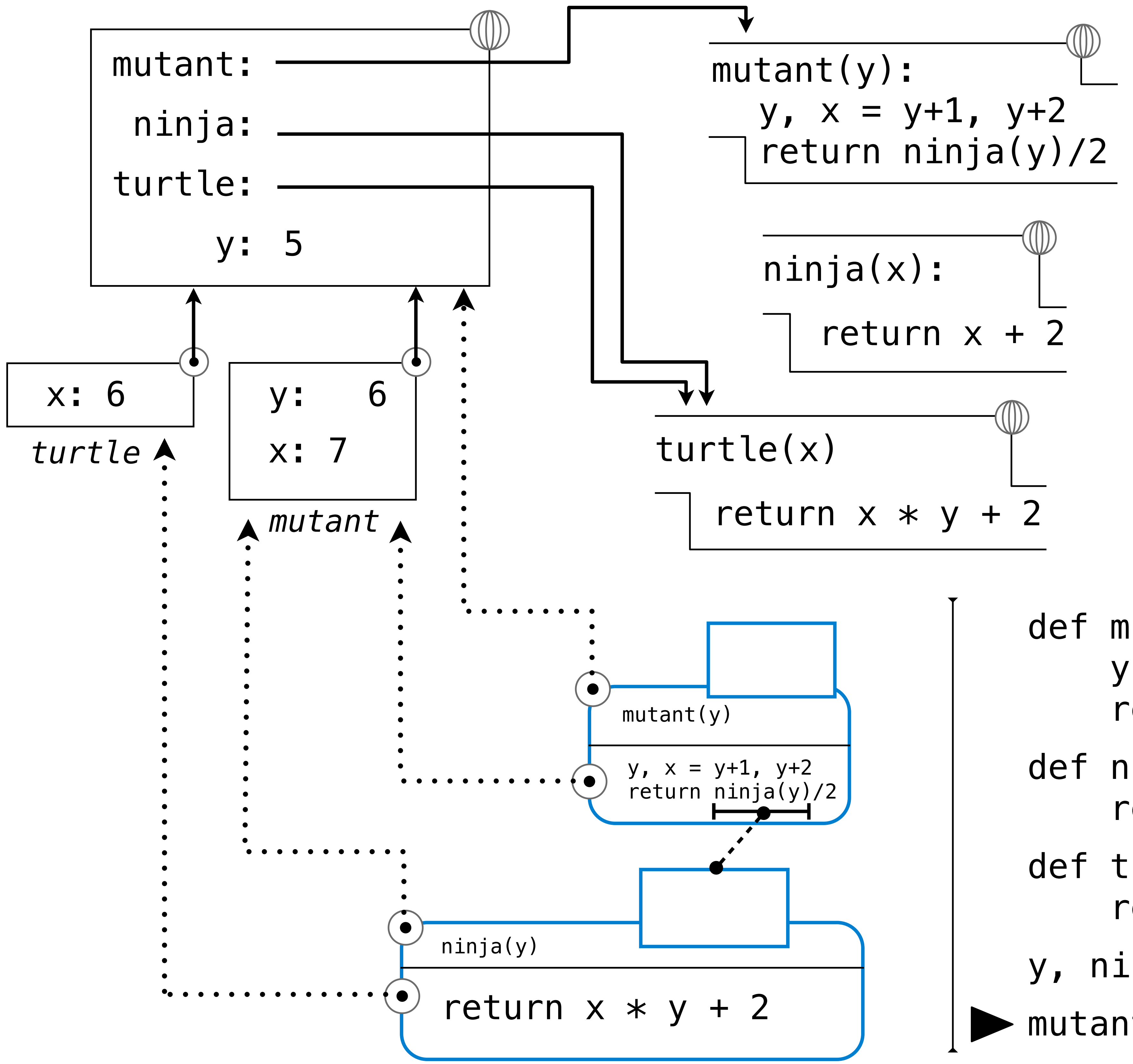
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



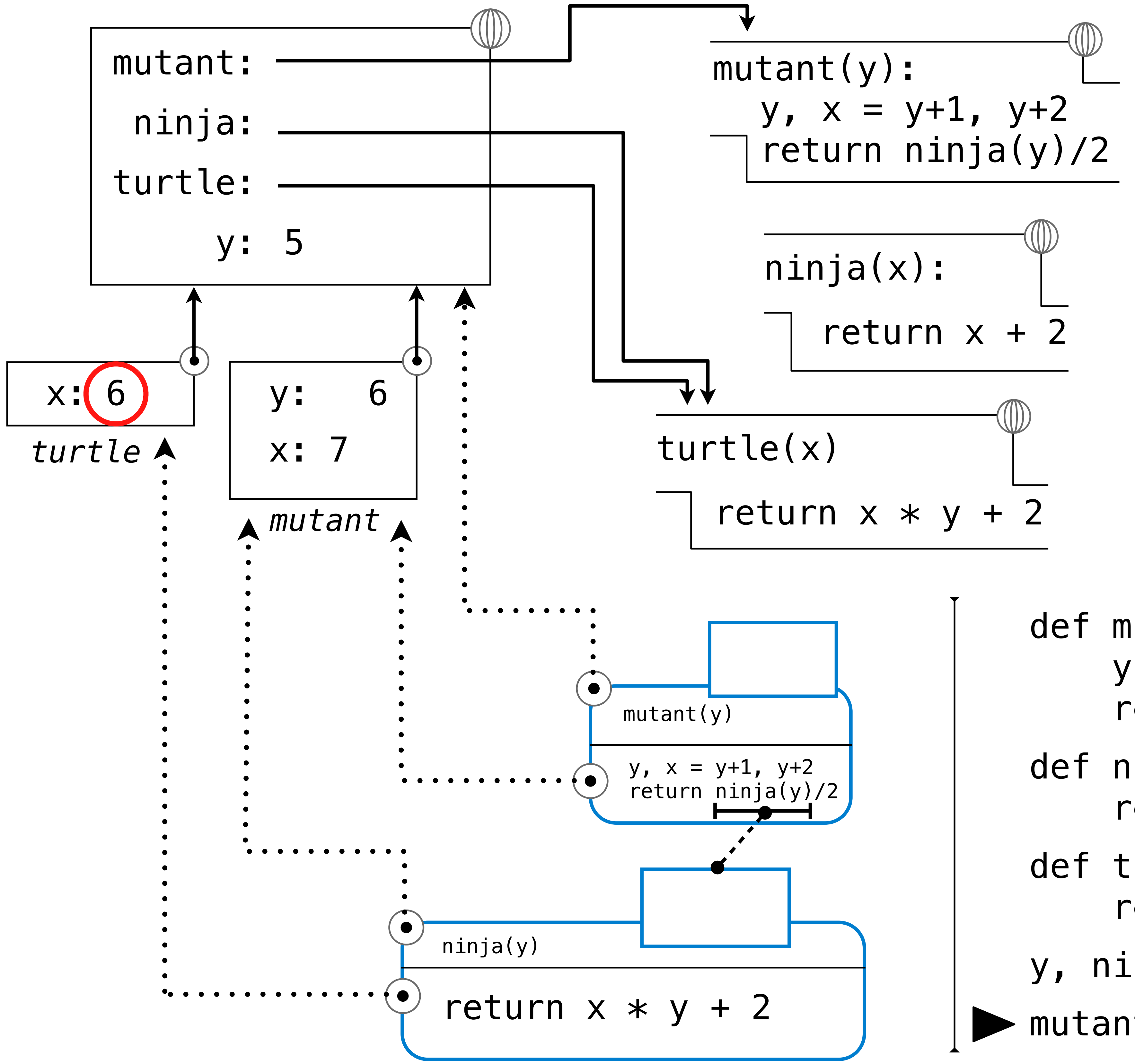
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

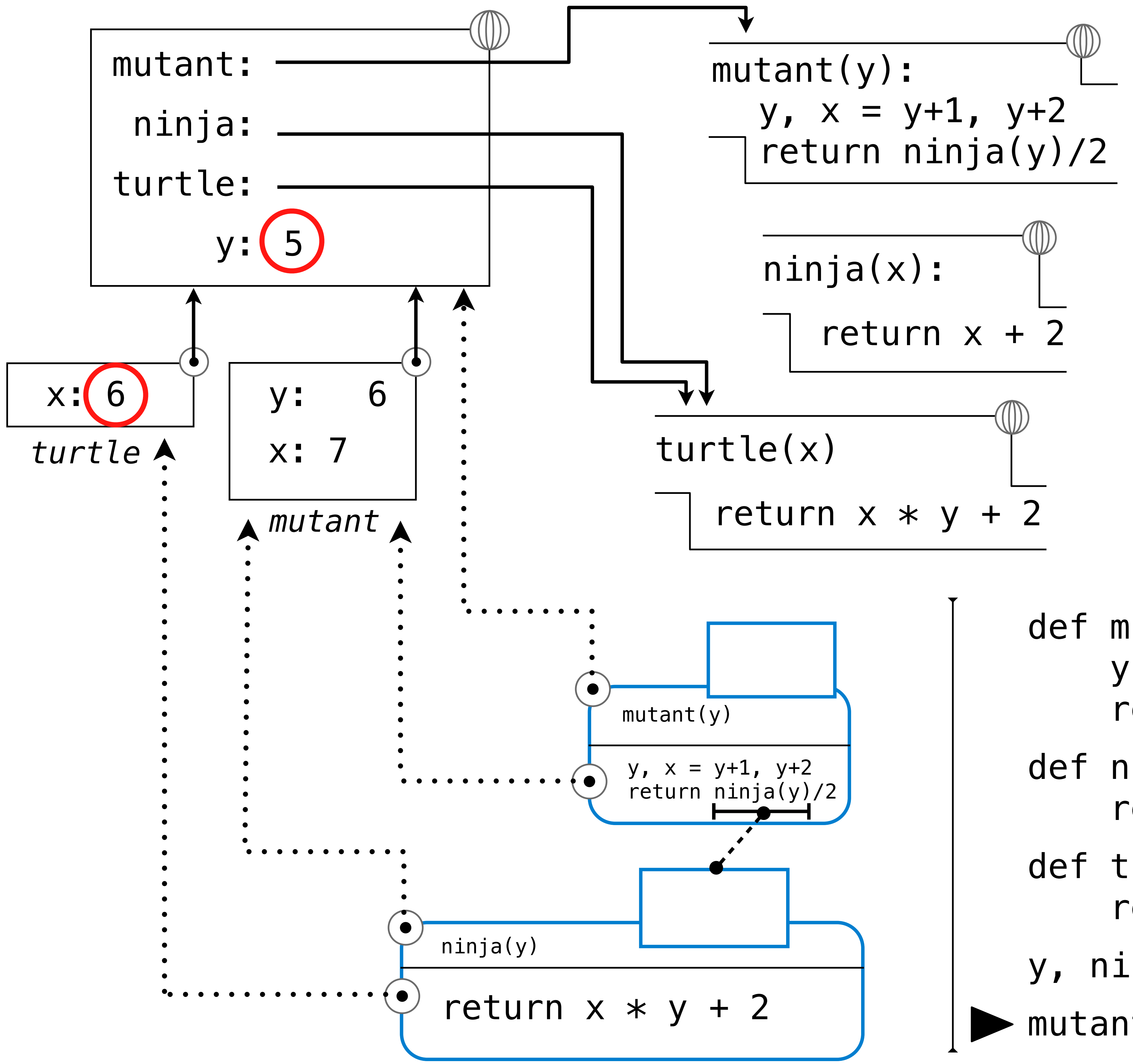
Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

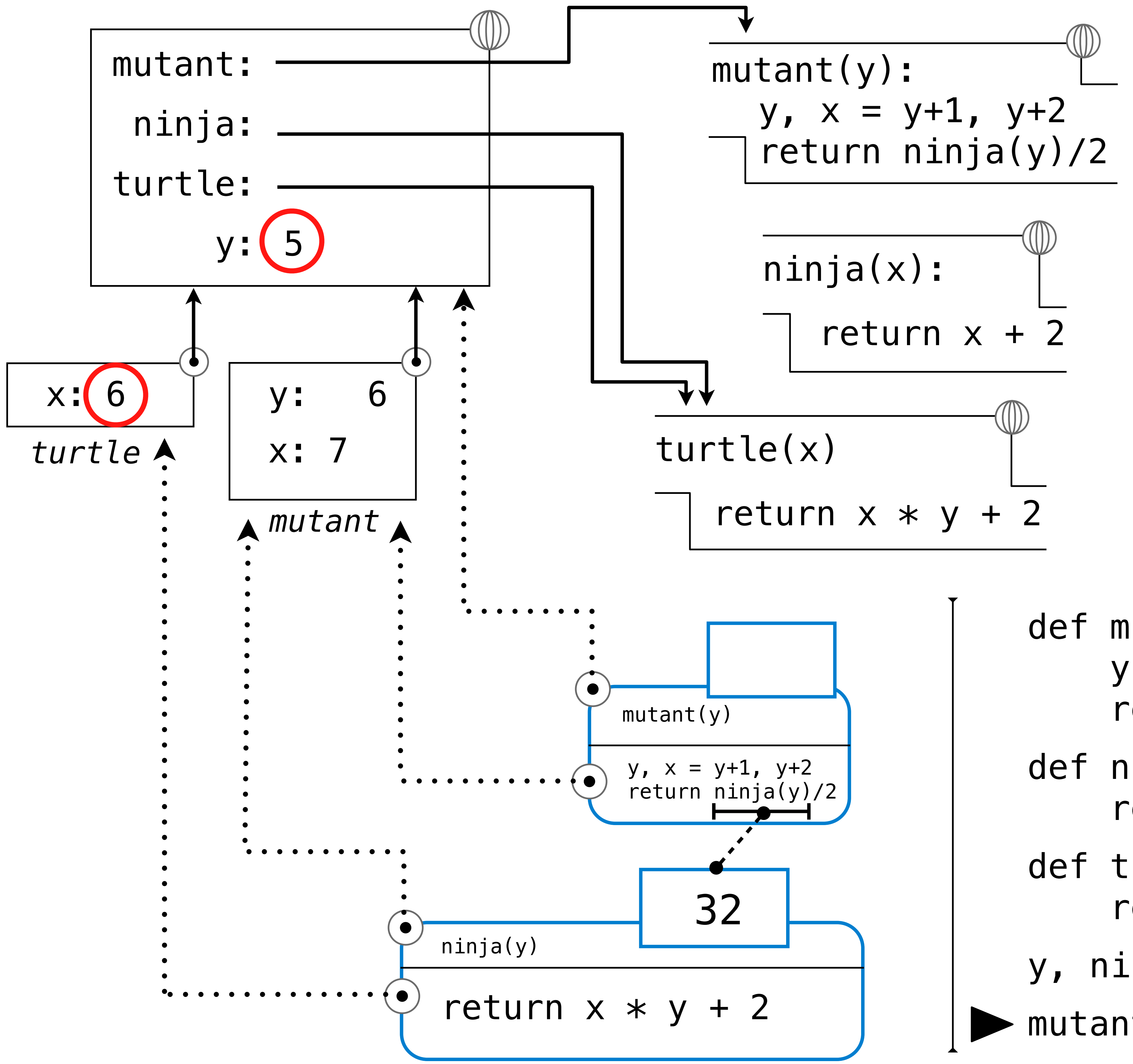
Friday, September 23, 2011

Assignment Review: Teenage Mutant Ninja Turtles



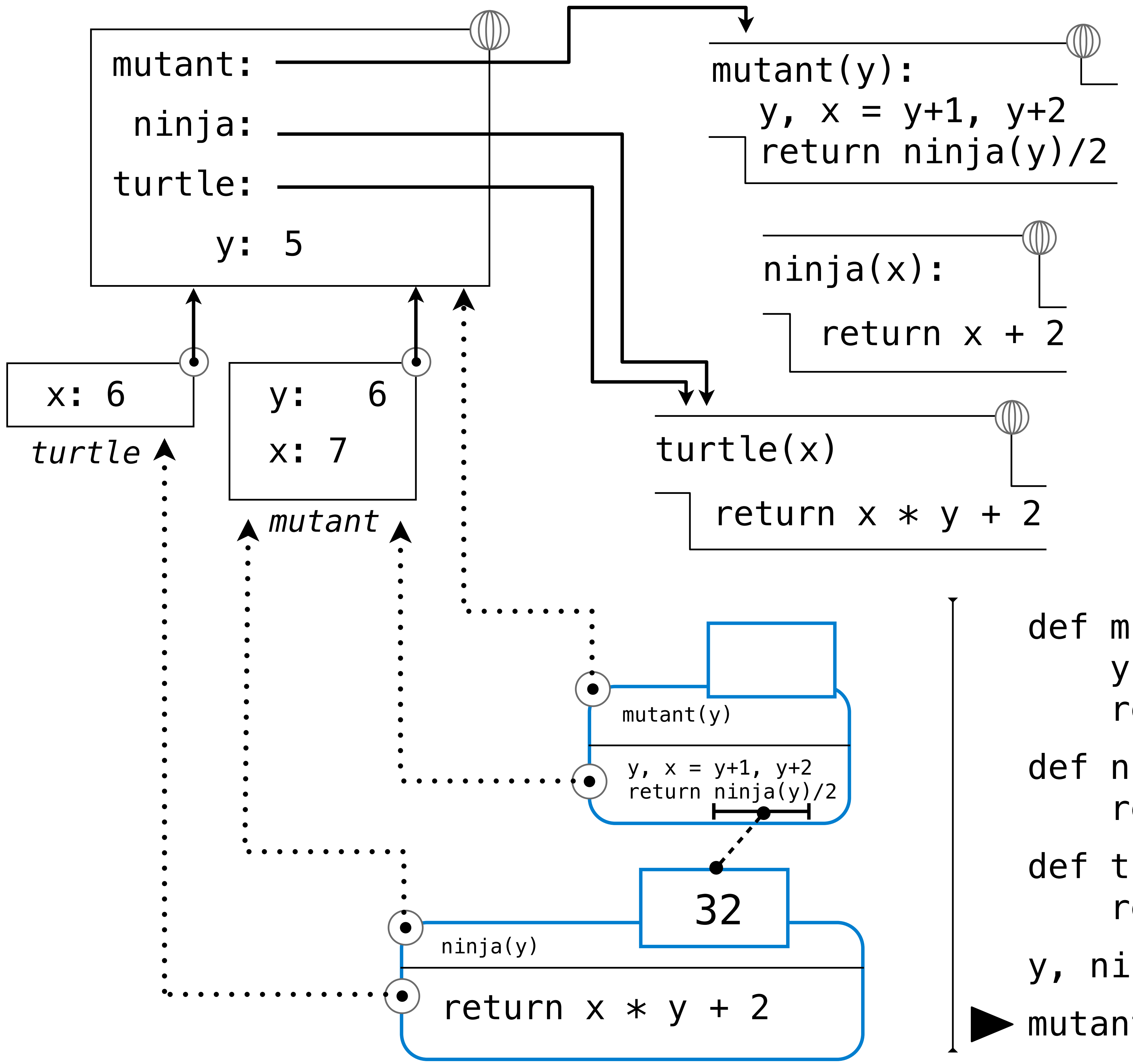
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



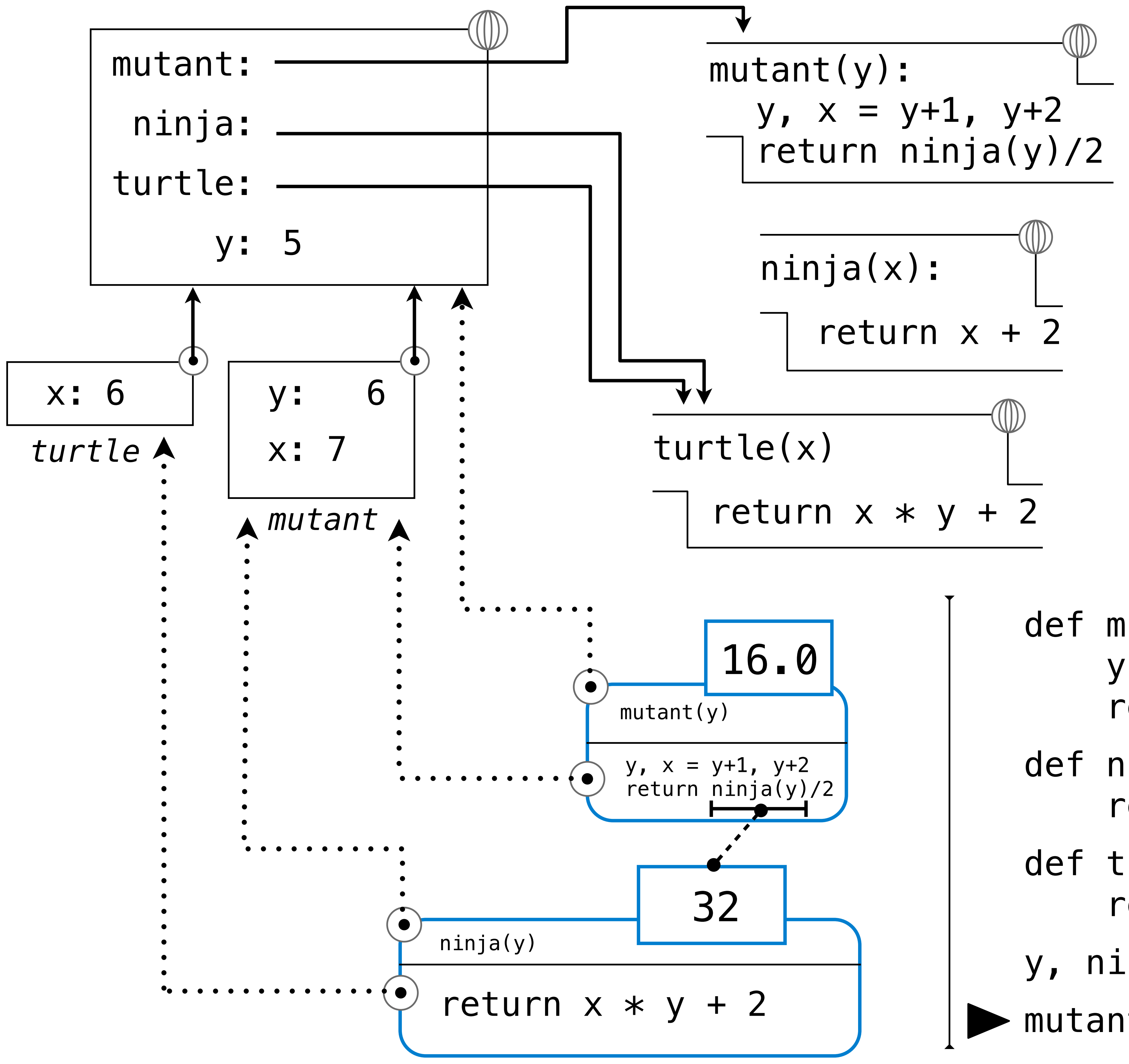
```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2
def ninja(x):
    return x + 2
def turtle(x):
    return x * y + 2
y, ninja = 5, turtle
mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles



```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function
  - Look up y, which is bound to 6

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function
  - Look up y, which is bound to 6
  - Apply the *turtle* function to 6

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function
  - Look up y, which is bound to 6
  - Apply the *turtle* function to 6
    - Look up x, which is bound to 6 in the local frame

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function
  - Look up y, which is bound to 6
  - Apply the *turtle* function to 6
    - Look up x, which is bound to 6 in the local frame
    - Look up y, which is bound to 5 in the global frame

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function
- Apply the *mutant* function *to 5*
    - In the first frame, bind y to 6 and x to 7
    - Look up ninja, which is bound to the *turtle* function
    - Look up y, which is bound to 6
    - Apply the *turtle* function to 6
        - Look up x, which is bound to 6 in the local frame
        - Look up y, which is bound to 5 in the global frame
        - Return 6 * 5 + 2 = 32

Intrinsic function name

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Assignment Review: Teenage Mutant Ninja Turtles

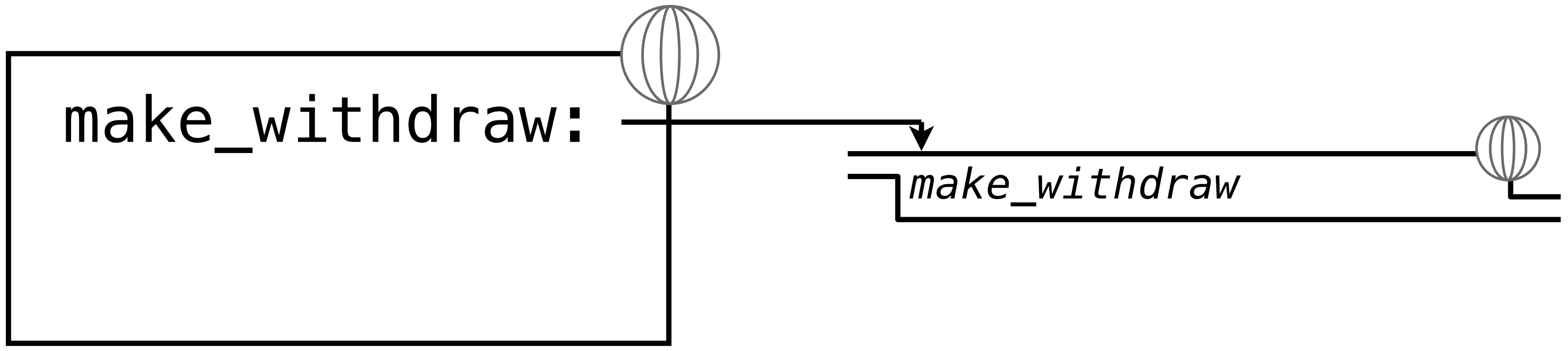
- Bind mutant, ninja, and turtle to their respective functions
- Simultaneously: bind y to 5 and ninja to the *turtle* function

Intrinsic function name

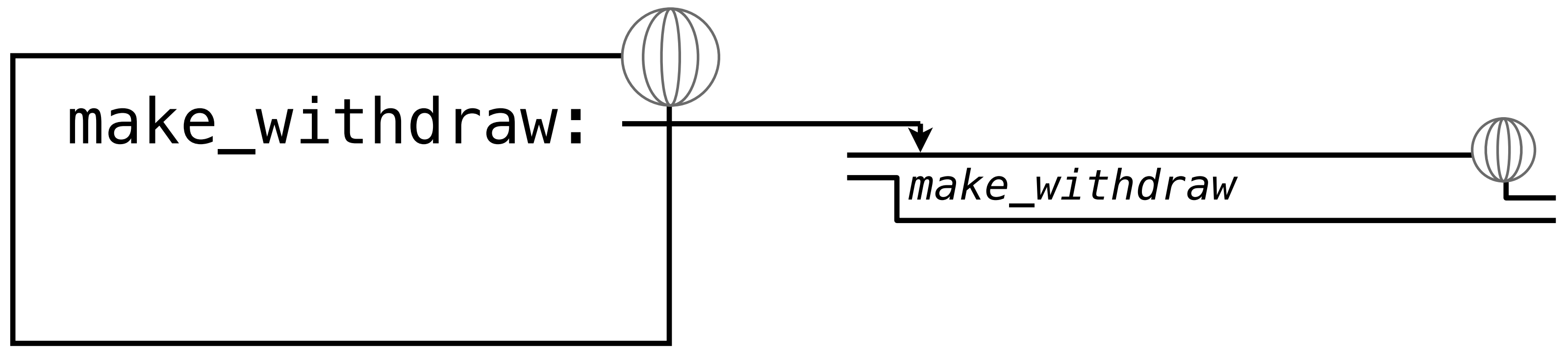
- Apply the *mutant* function *to 5*
  - In the first frame, bind y to 6 and x to 7
  - Look up ninja, which is bound to the *turtle* function
  - Look up y, which is bound to 6
  - Apply the *turtle* function to 6
    - Look up x, which is bound to 6 in the local frame
    - Look up y, which is bound to 5 in the global frame
    - Return 6 * 5 + 2 = 32
  - Return 32 / 2 = 16.0

```
def mutant(y):
    y, x = y+1, y+2
    return ninja(y)/2

def ninja(x):
    return x + 2

def turtle(x):
    return x * y + 2

y, ninja = 5, turtle

mutant(y)
```

Environment Diagram of Withdraw

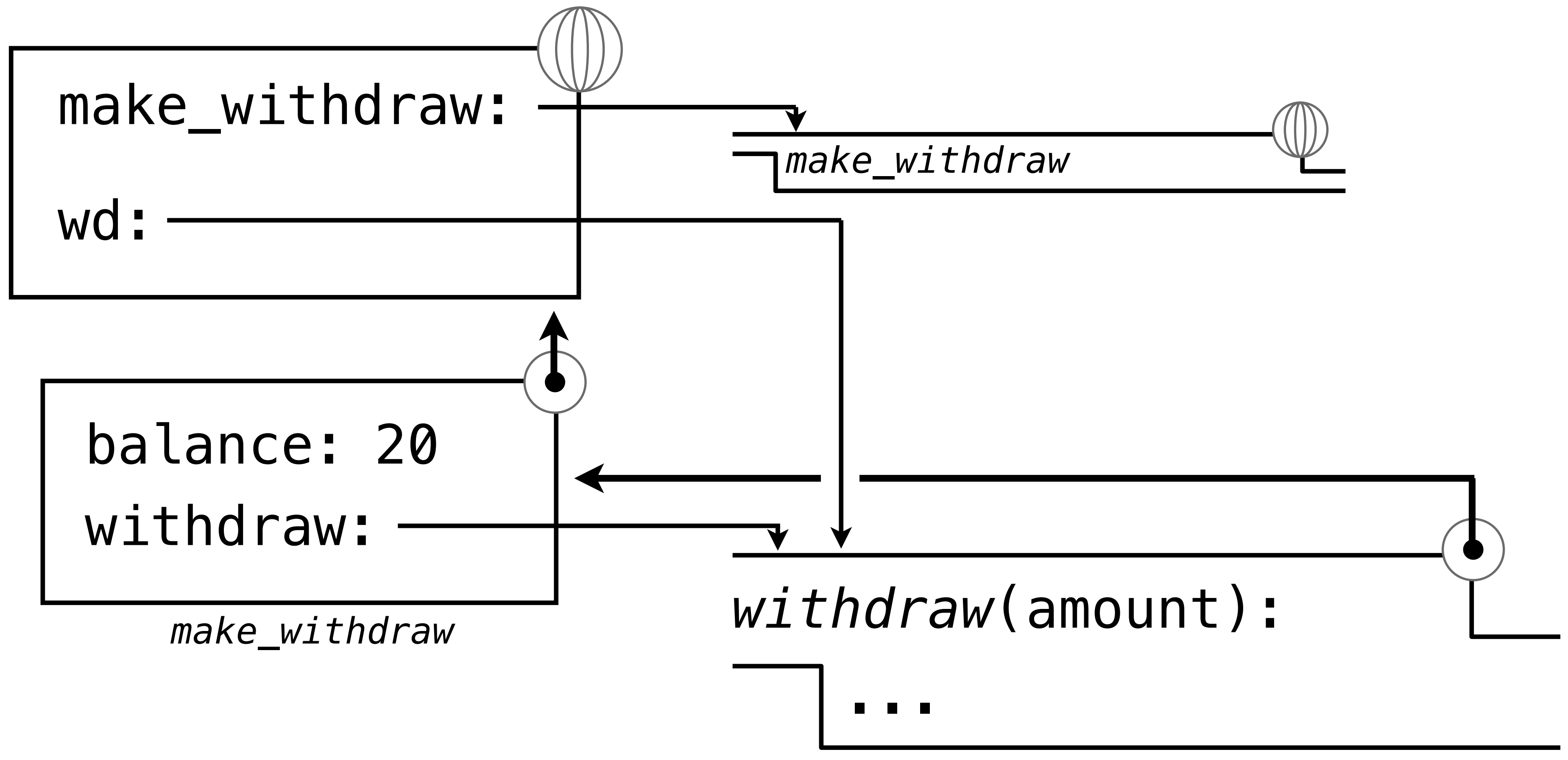


Environment Diagram of Withdraw



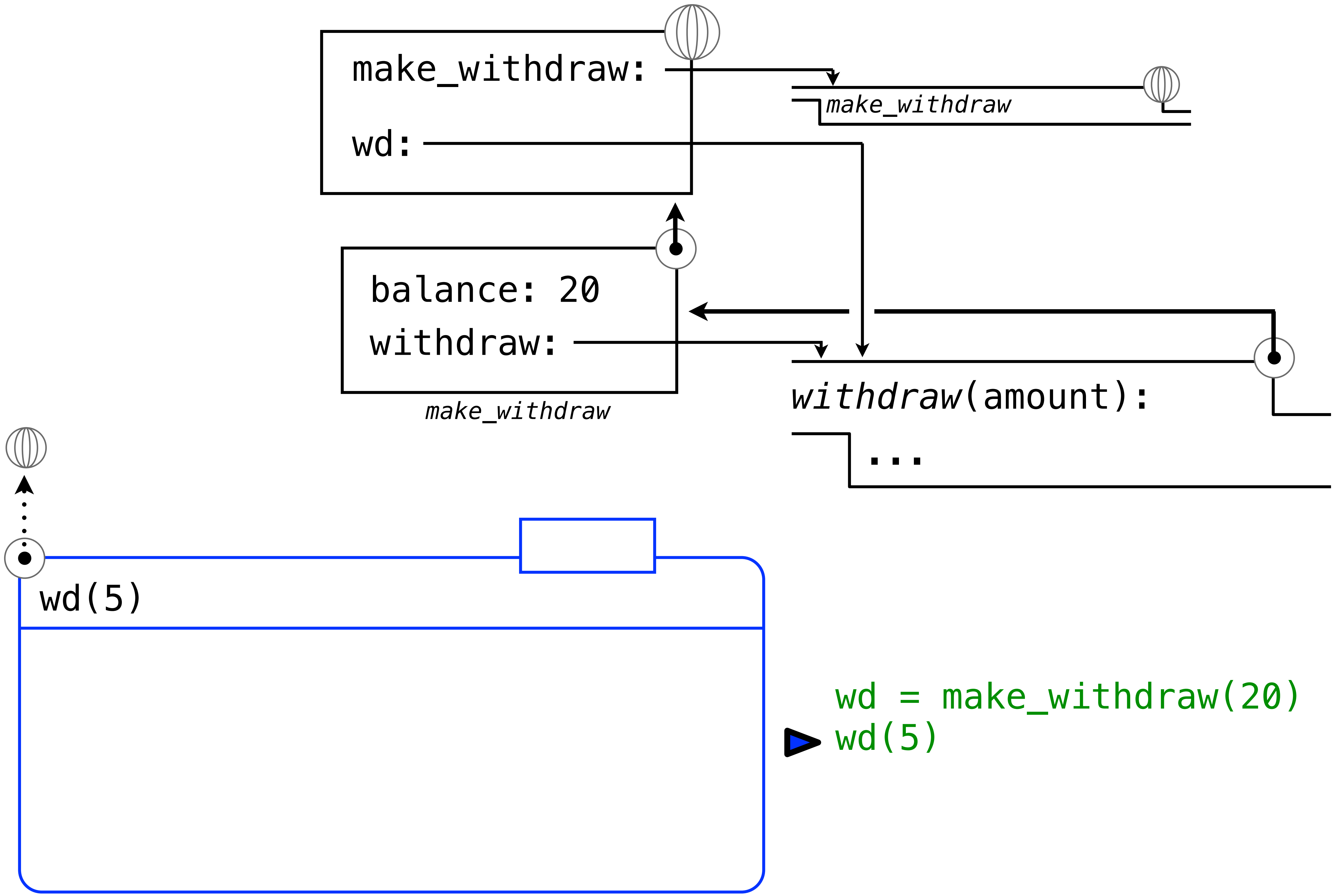
```
wd = make_withdraw(20)
wd(5)
```

Environment Diagram of Withdraw



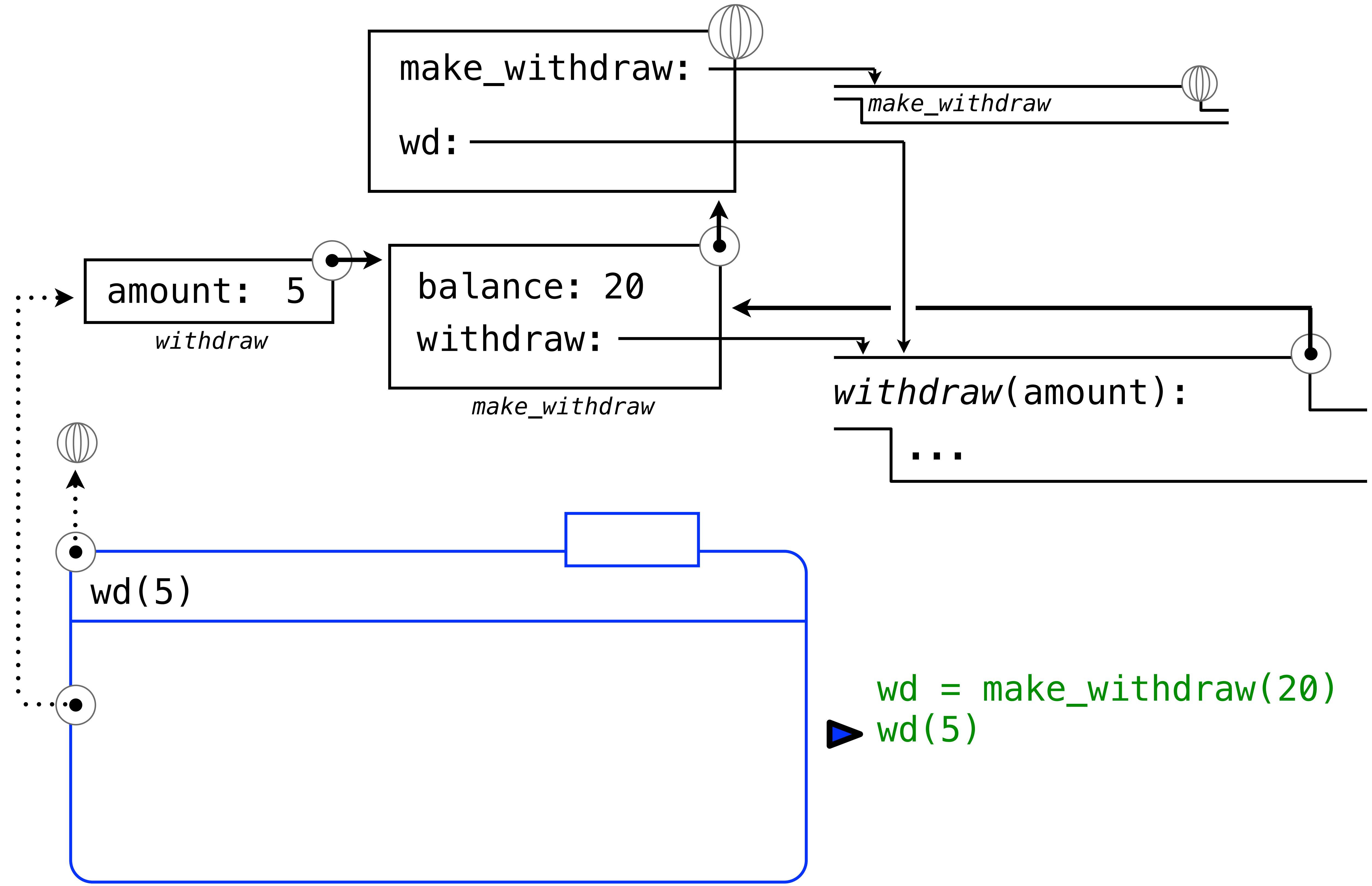
```
wd = make_withdraw(20)
wd(5)
```

Environment Diagram of Withdraw



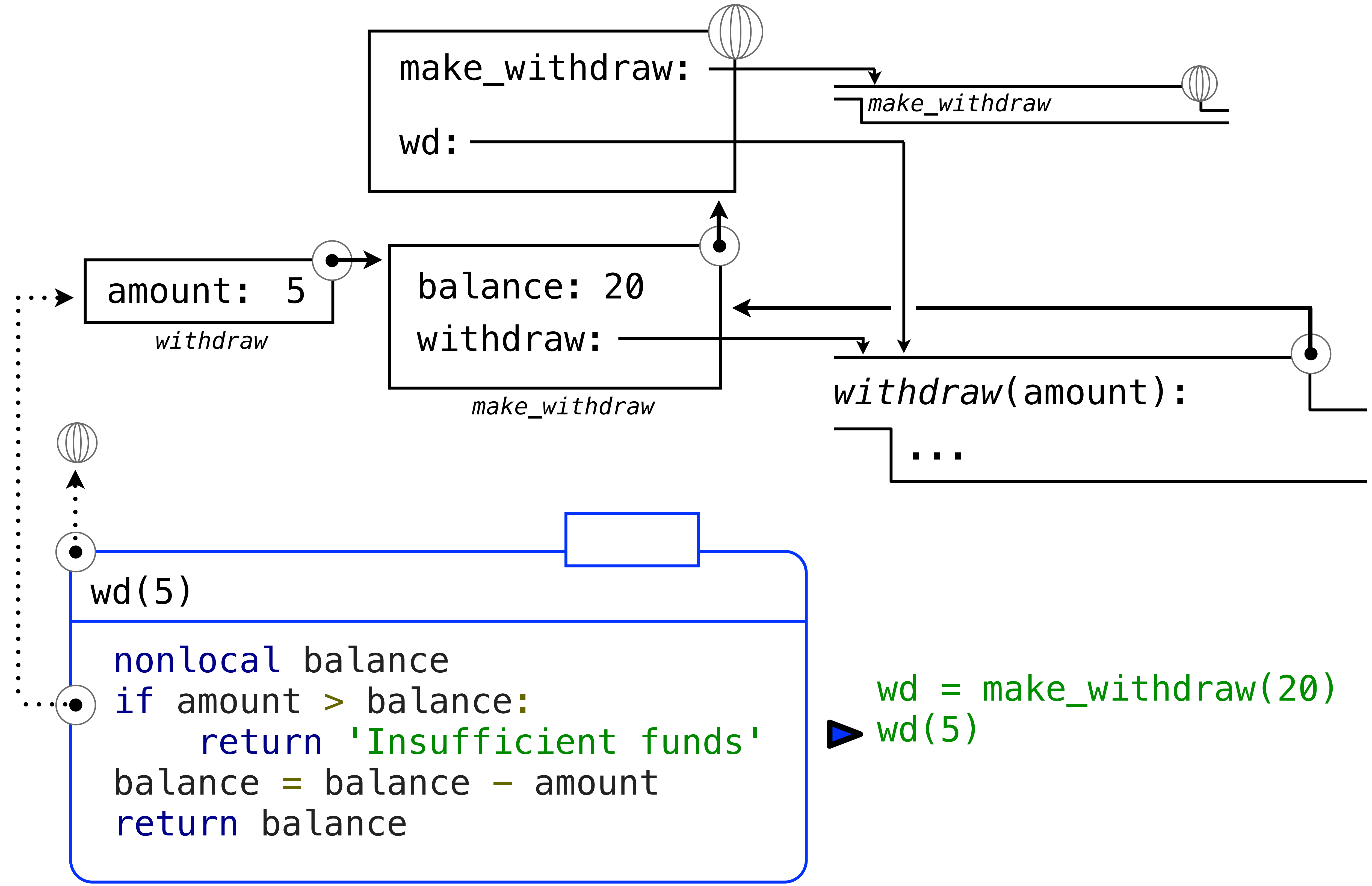
```
wd = make_withdraw(20)
wd(5)
```

Environment Diagram of Withdraw



```
wd = make_withdraw(20)
wd(5)
```

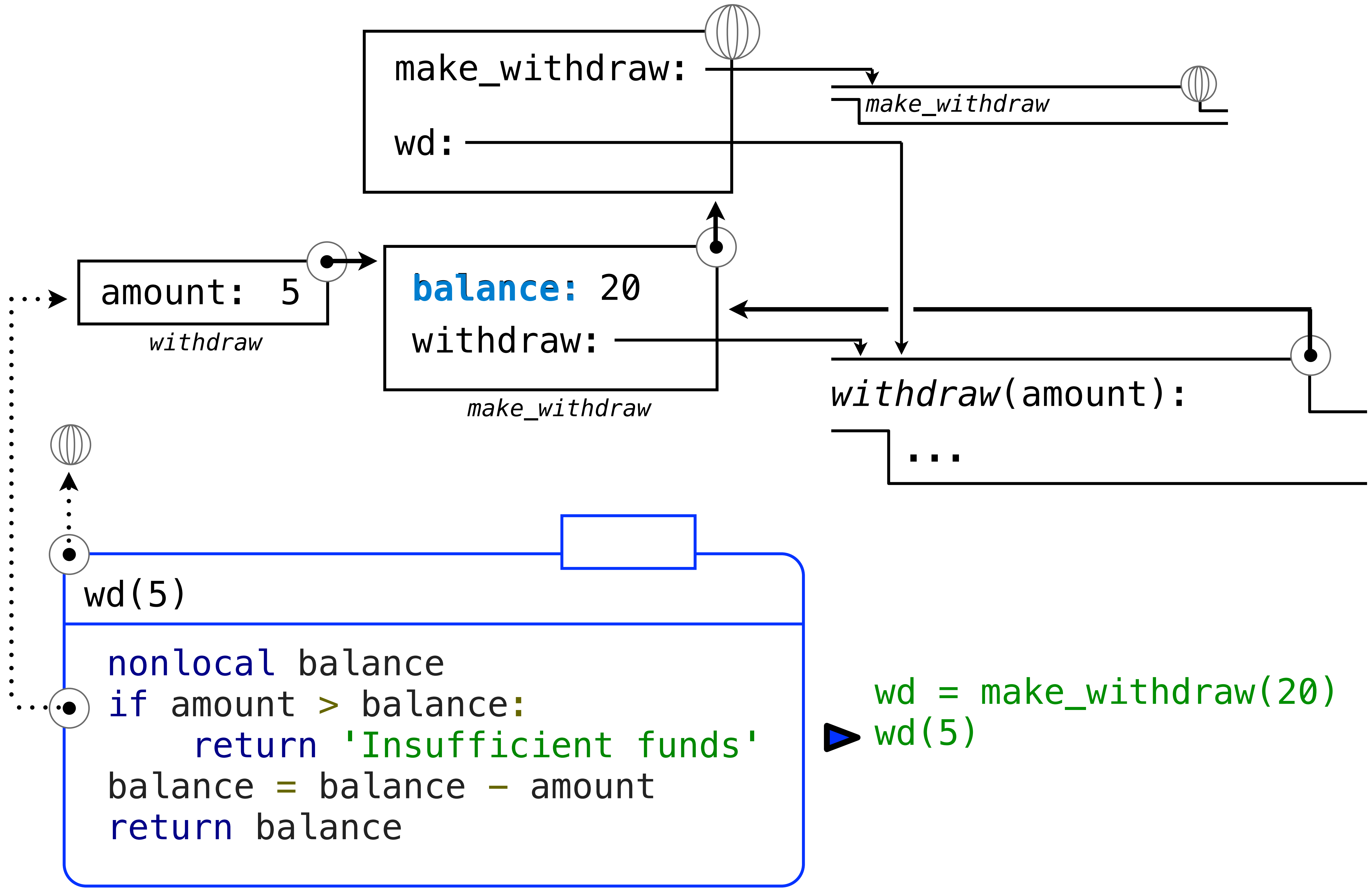
Environment Diagram of Withdraw



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
```

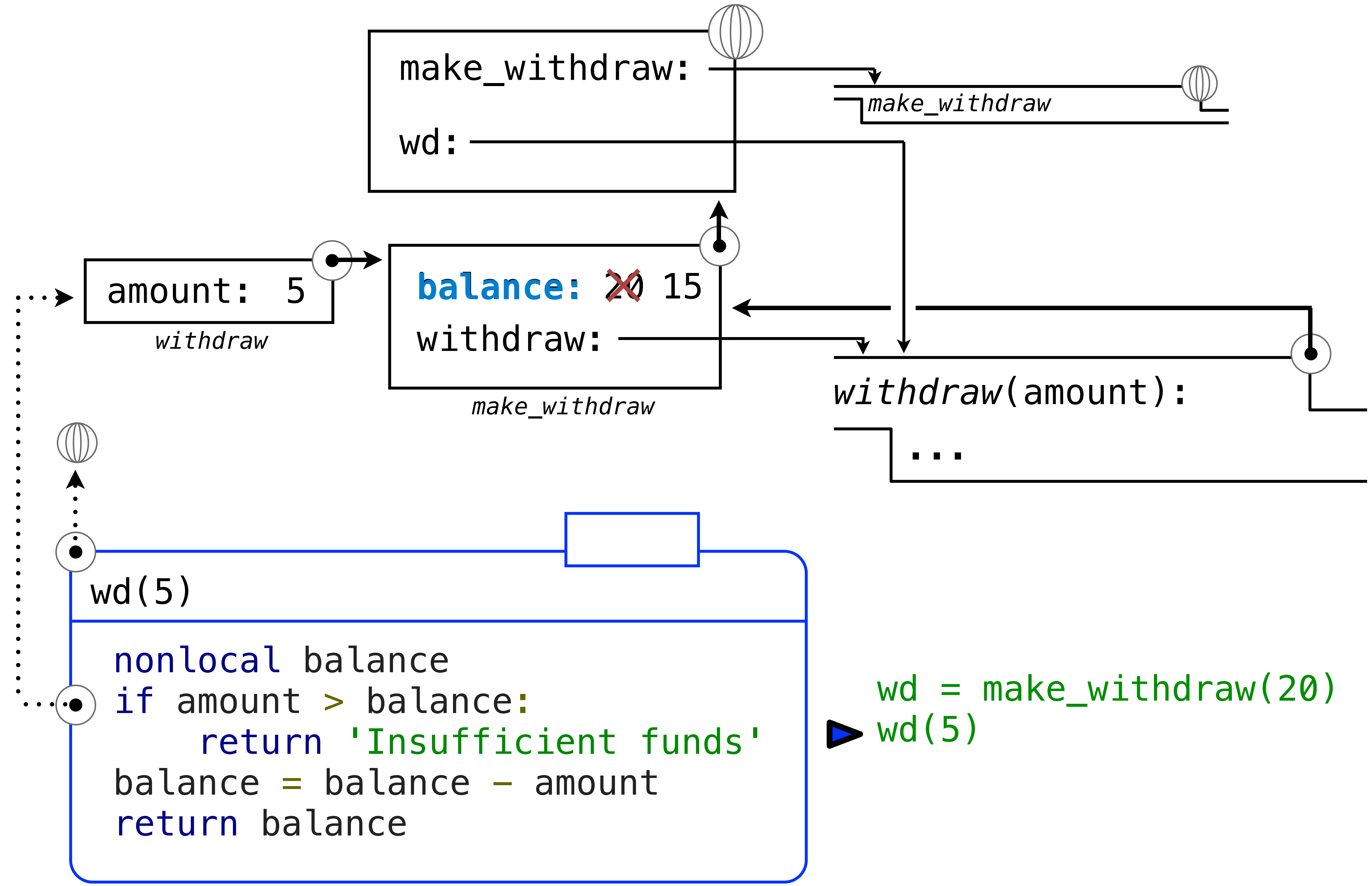
Environment Diagram of Withdraw



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
```

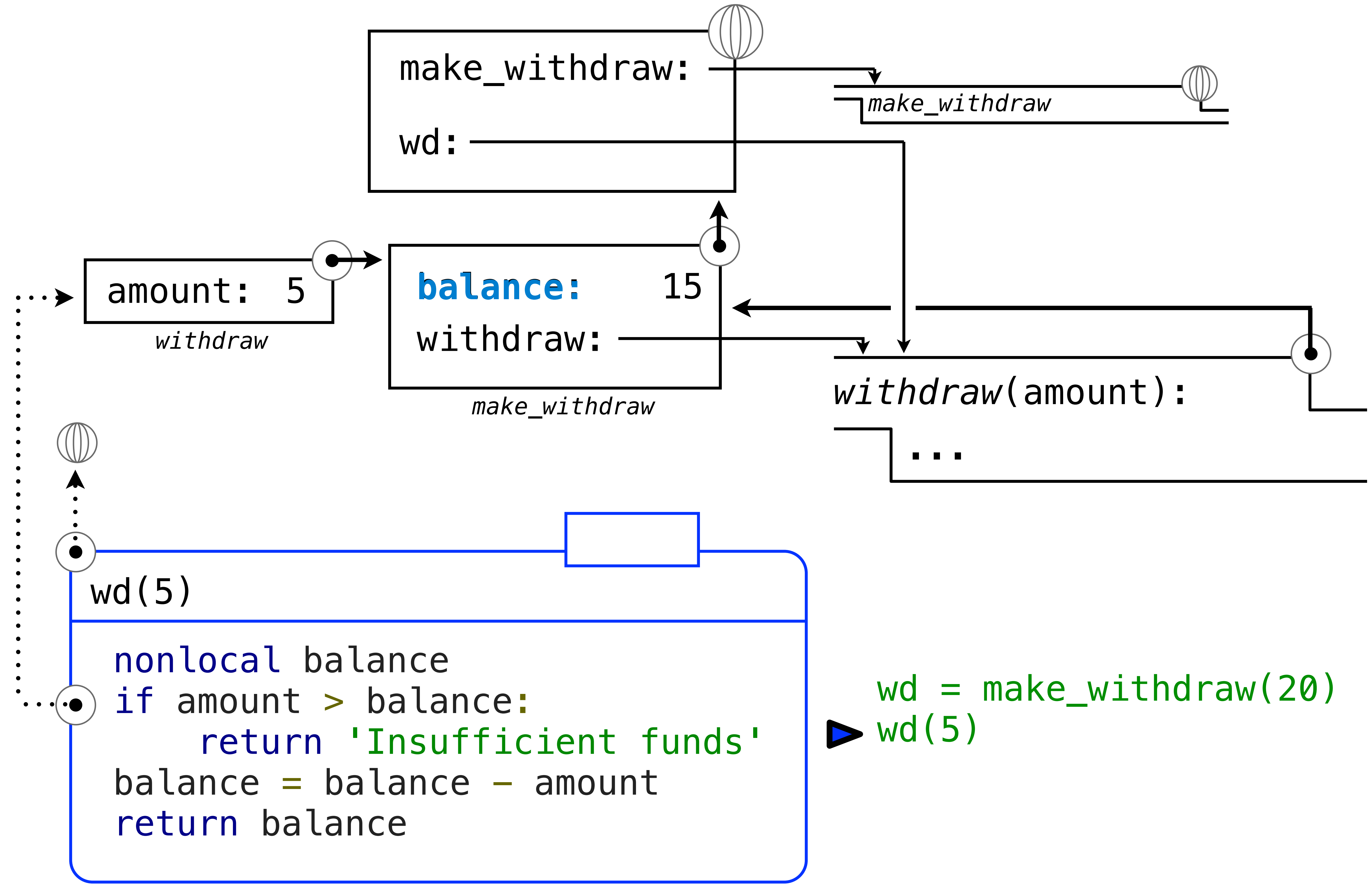
Environment Diagram of Withdraw



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
```

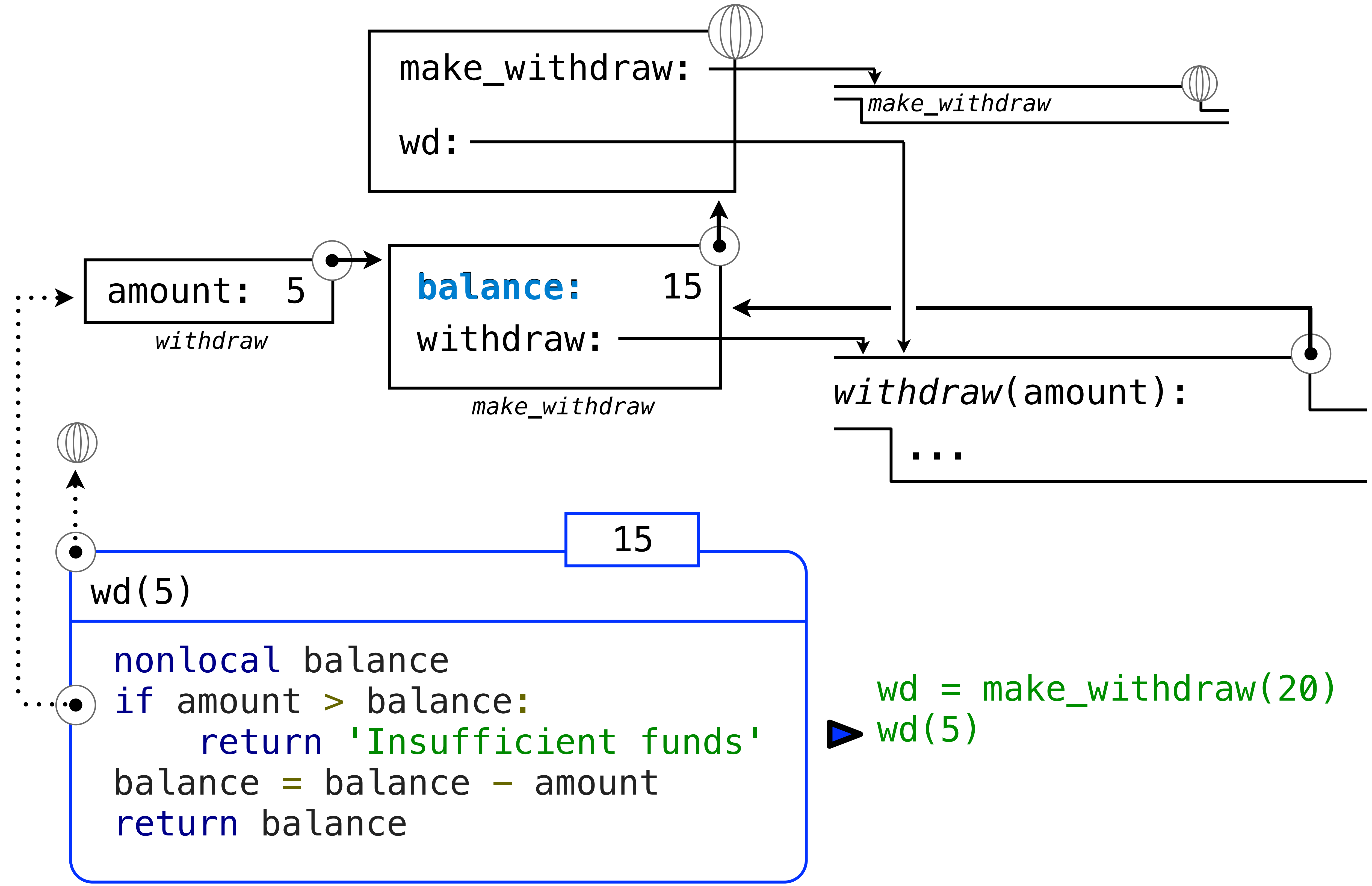
Environment Diagram of Withdraw



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
```

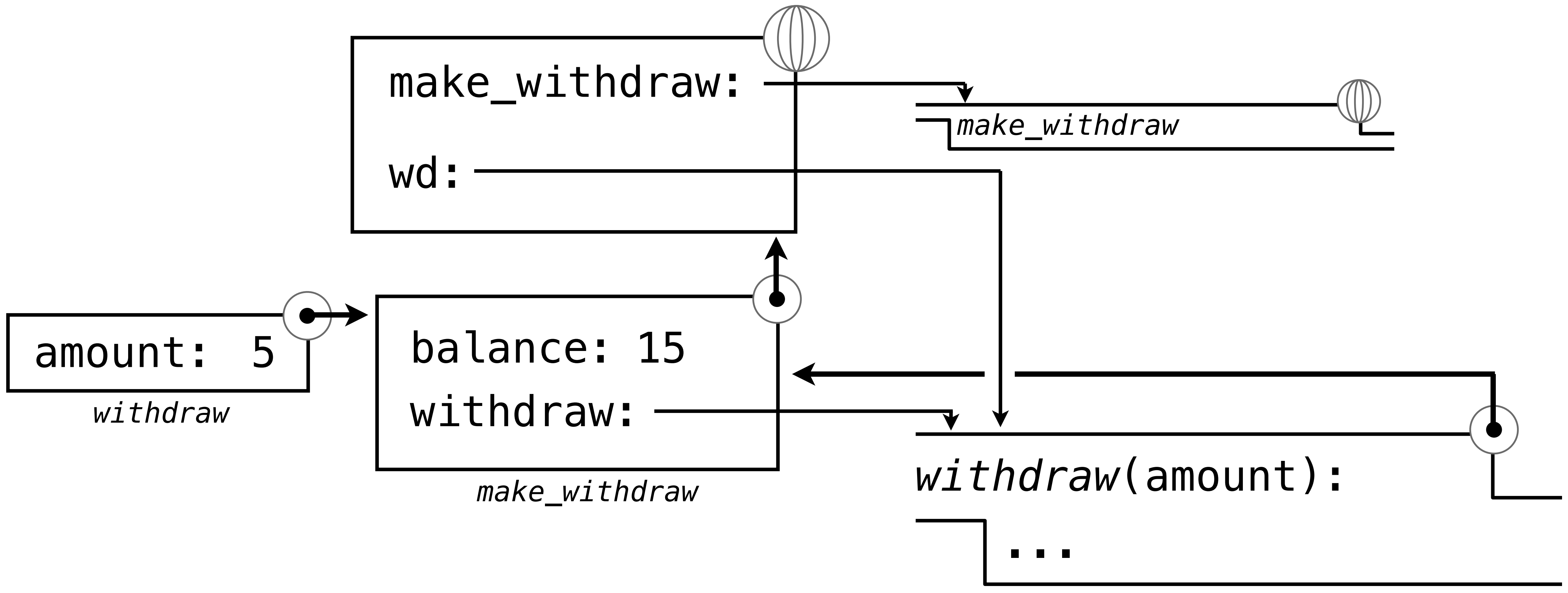
Environment Diagram of Withdraw



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

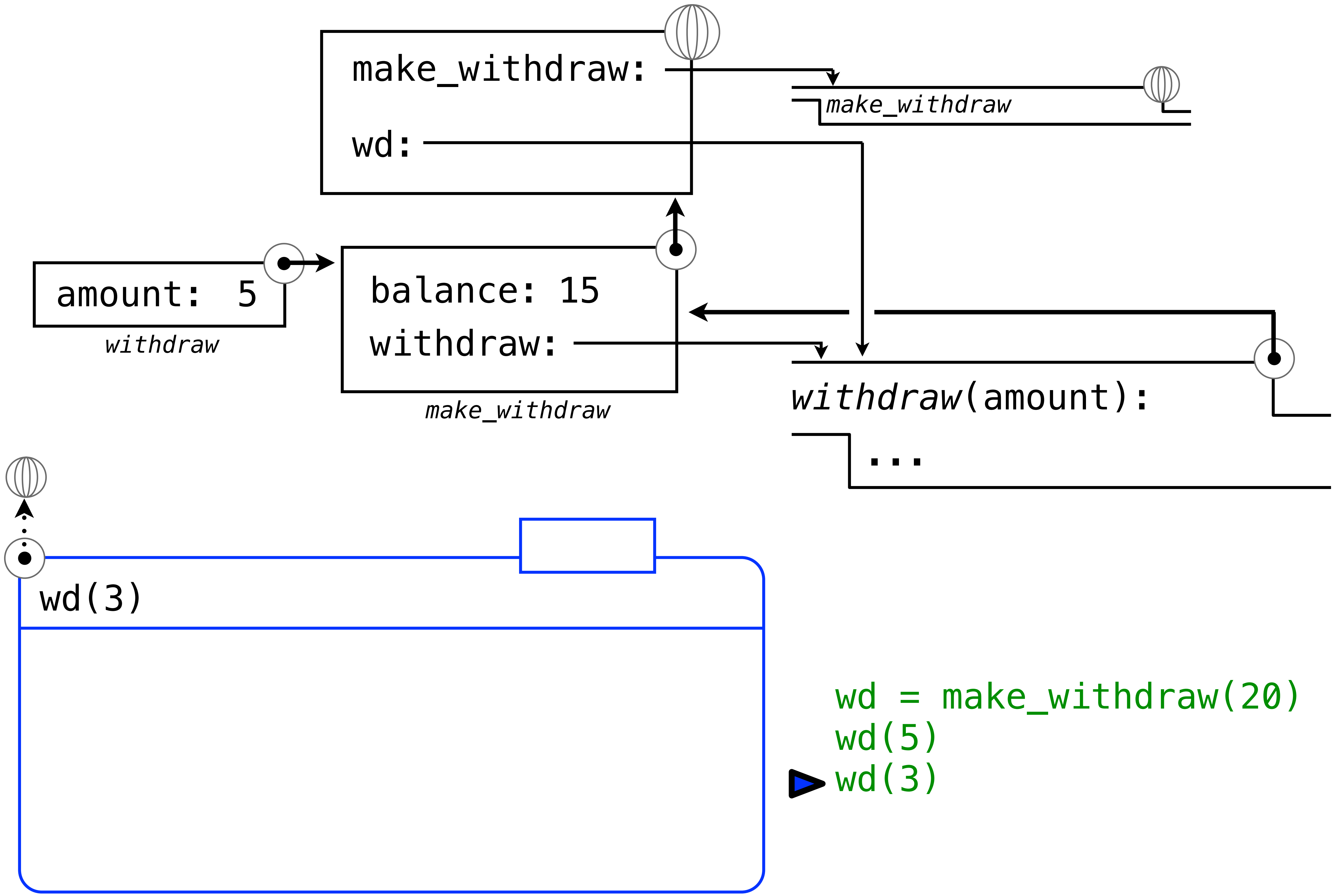
```
wd = make_withdraw(20)
wd(5)
```

Calling a Withdraw Function Twice



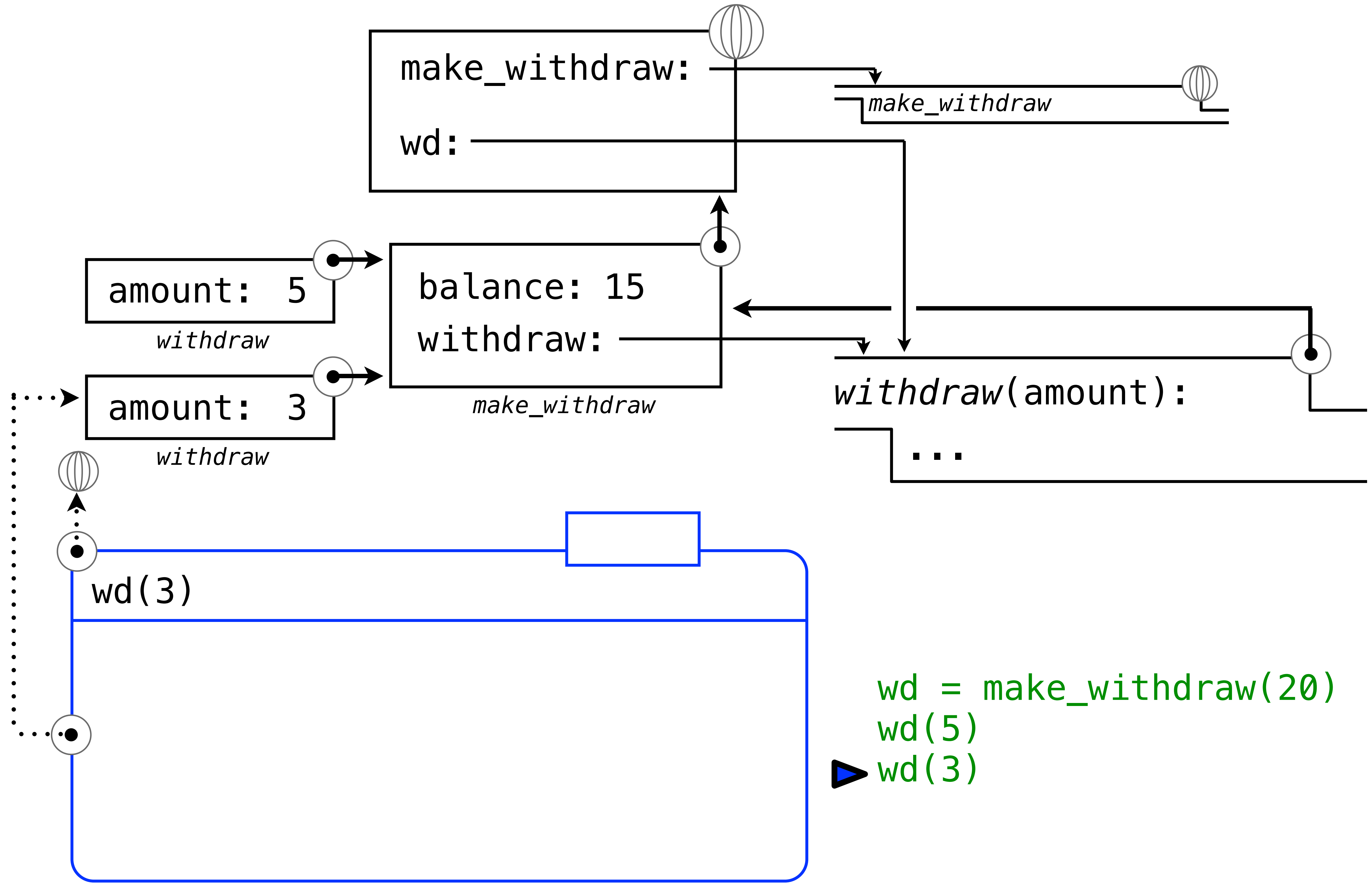
```
wd = make_withdraw(20)
wd(5)
wd(3)
```

Calling a Withdraw Function Twice



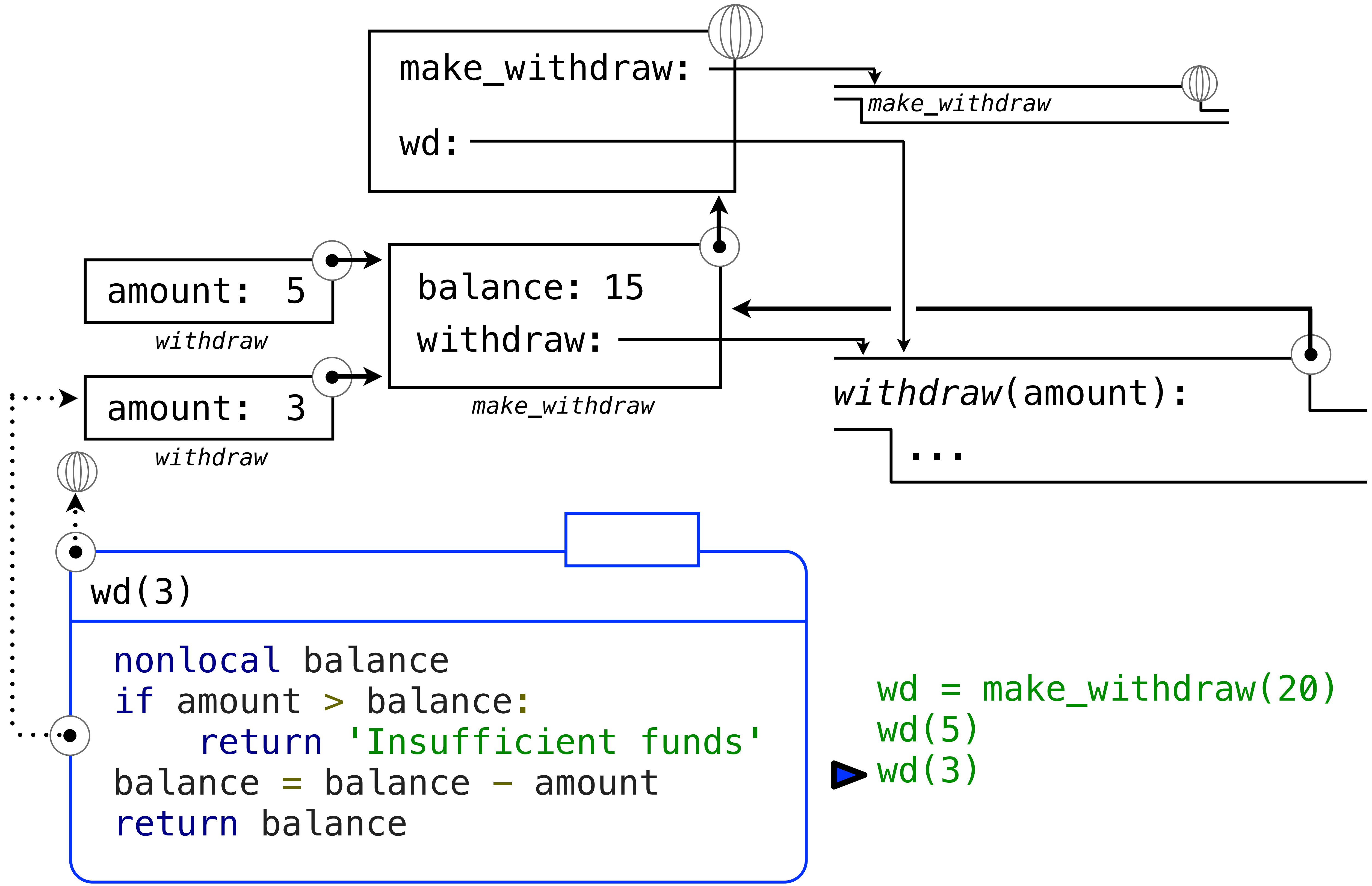
```
wd = make_withdraw(20)
wd(5)
wd(3)
```

Calling a Withdraw Function Twice



```
wd = make_withdraw(20)
wd(5)
wd(3)
```

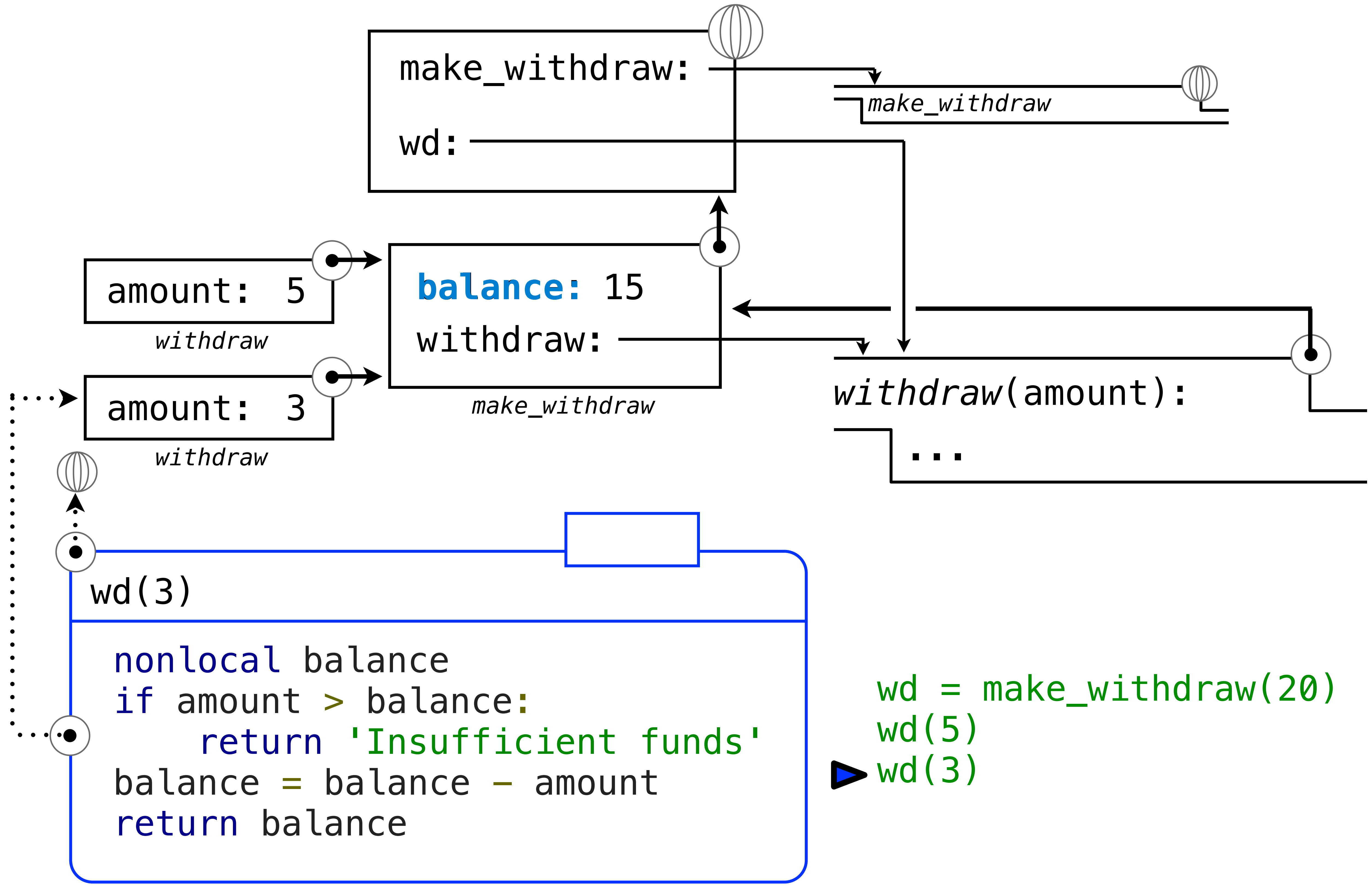
Calling a Withdraw Function Twice



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

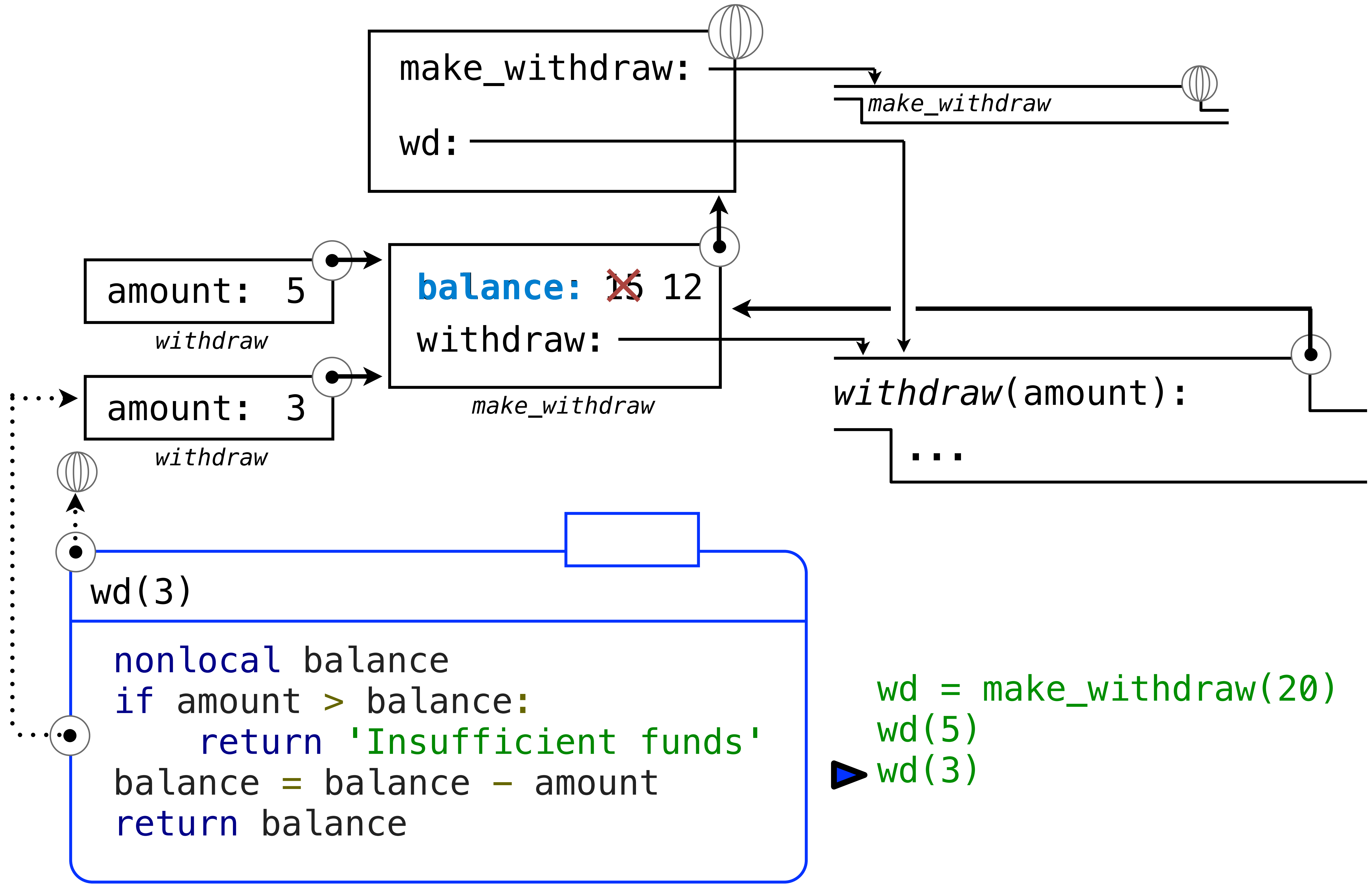
Calling a Withdraw Function Twice



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

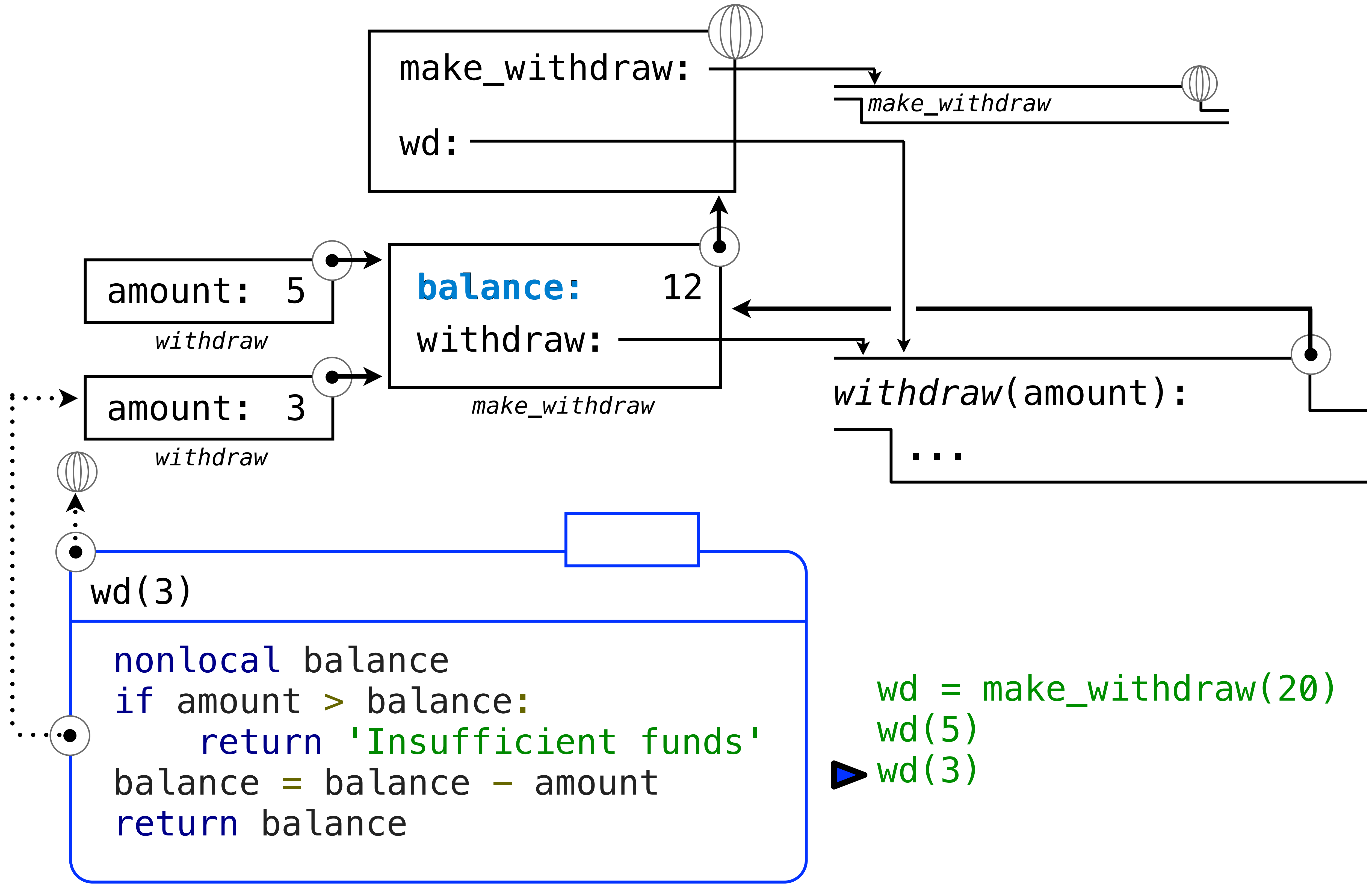
Calling a Withdraw Function Twice



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

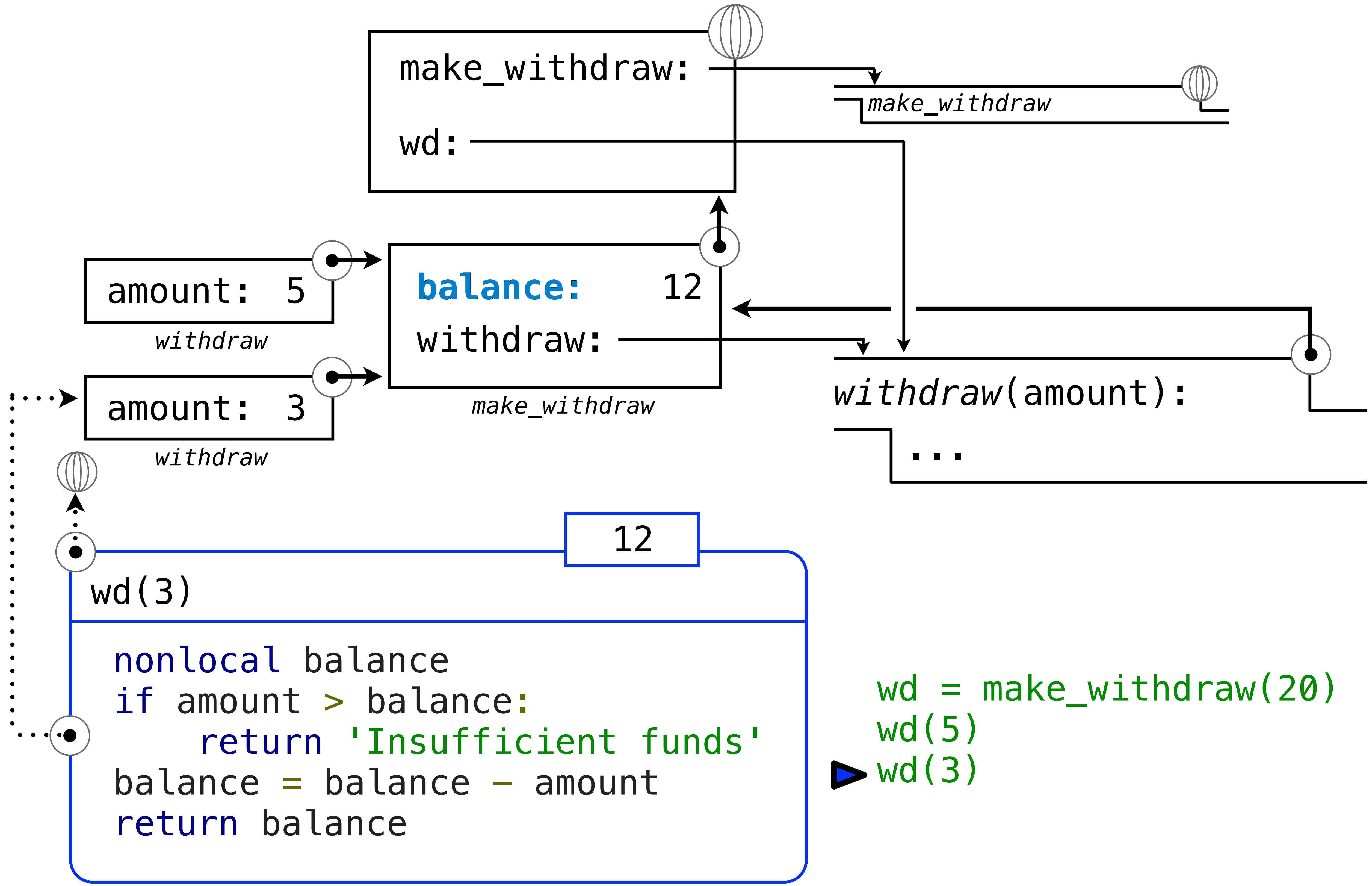
Calling a Withdraw Function Twice



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
```

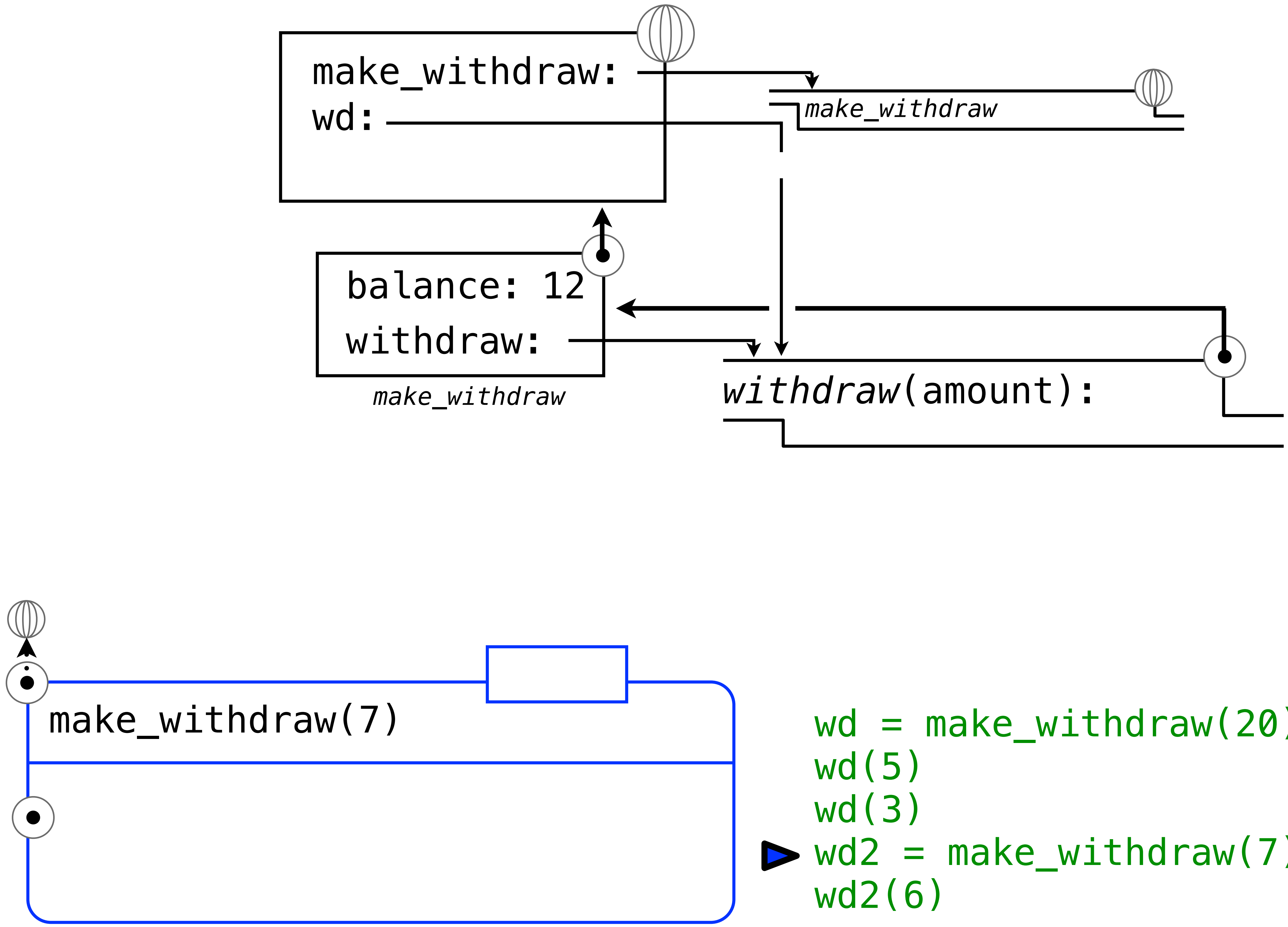
Calling a Withdraw Function Twice



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

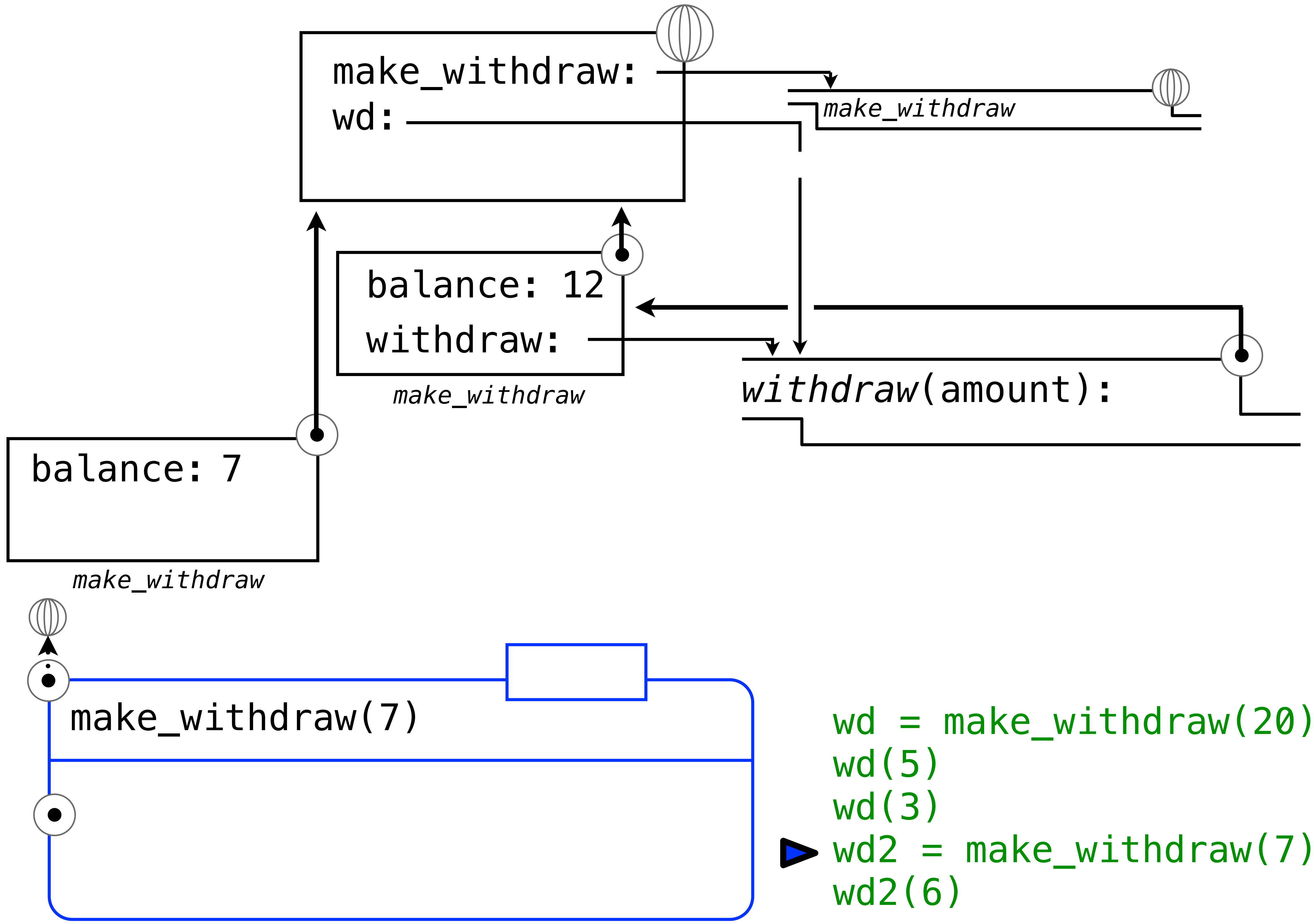
```
wd = make_withdraw(20)
wd(5)
wd(3)
```

Creating Two Different Withdraw Functions



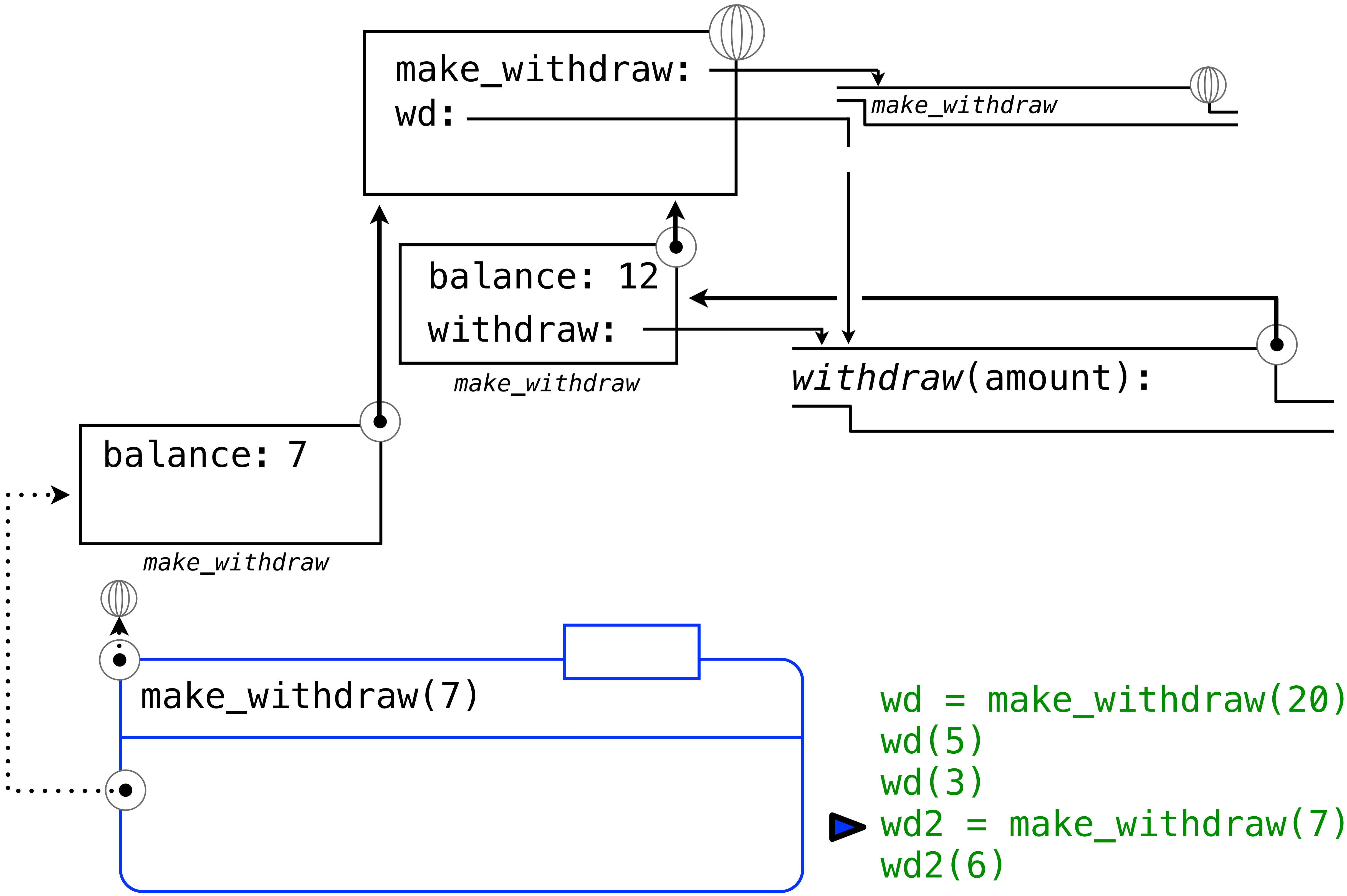
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



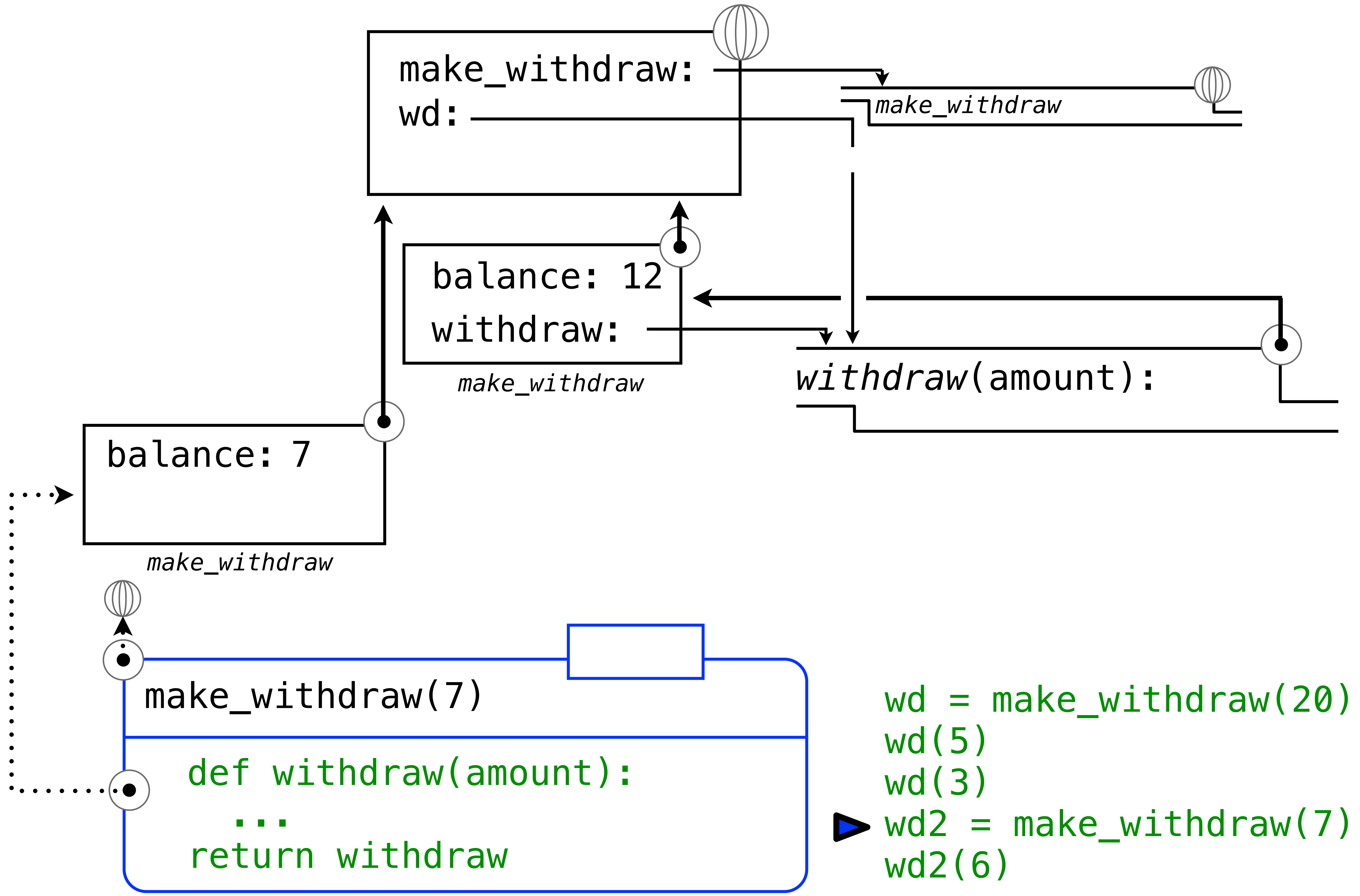
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



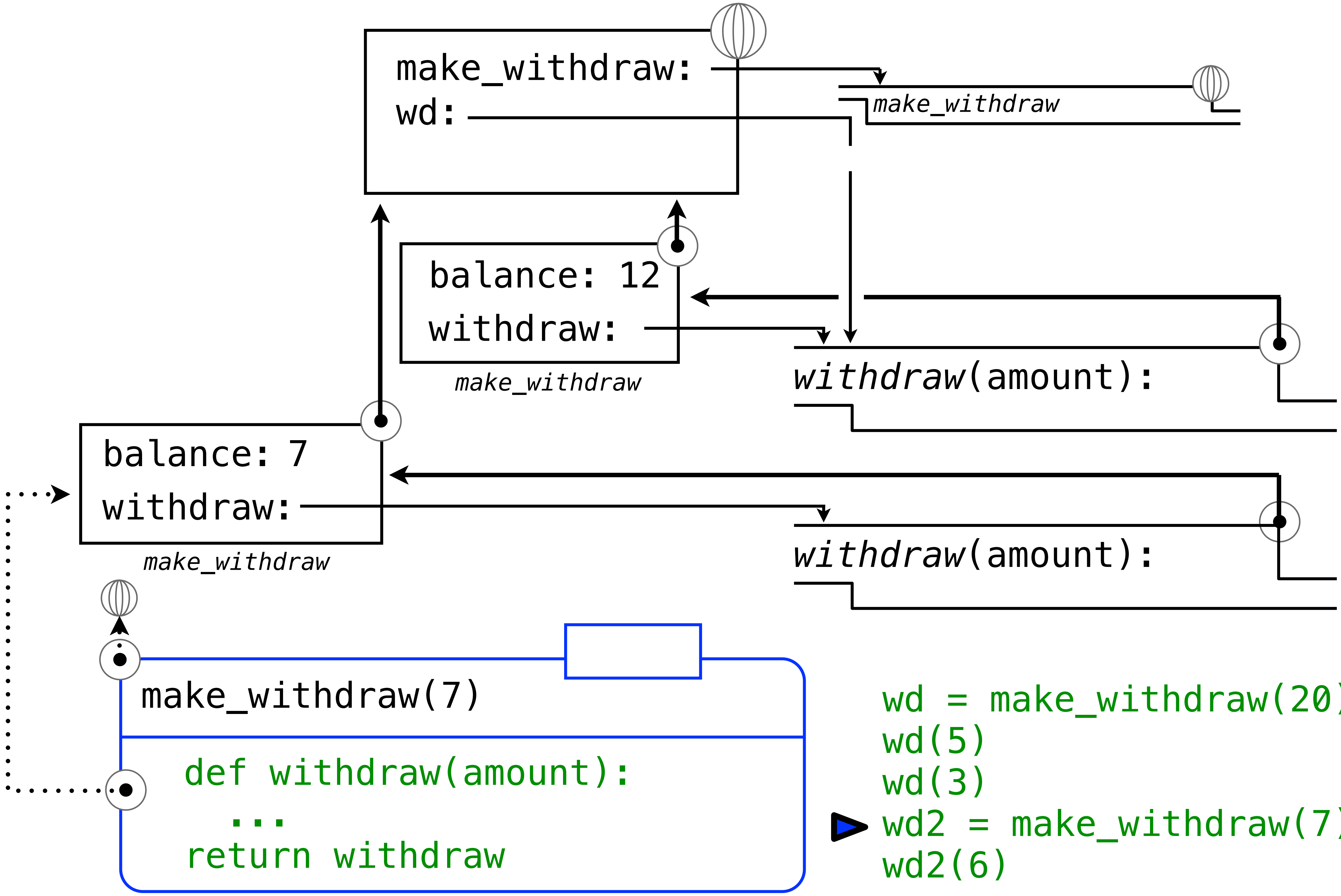
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



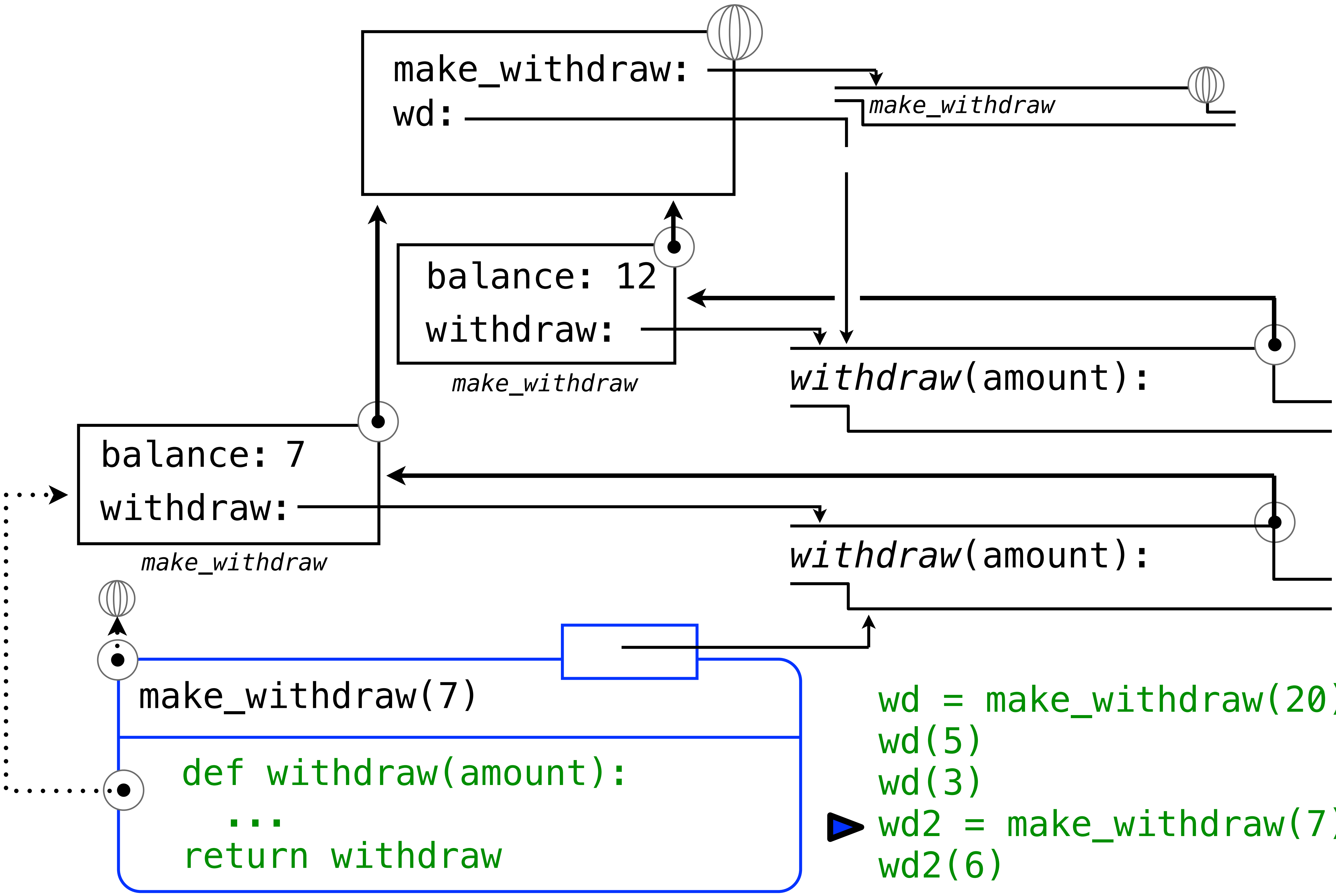
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



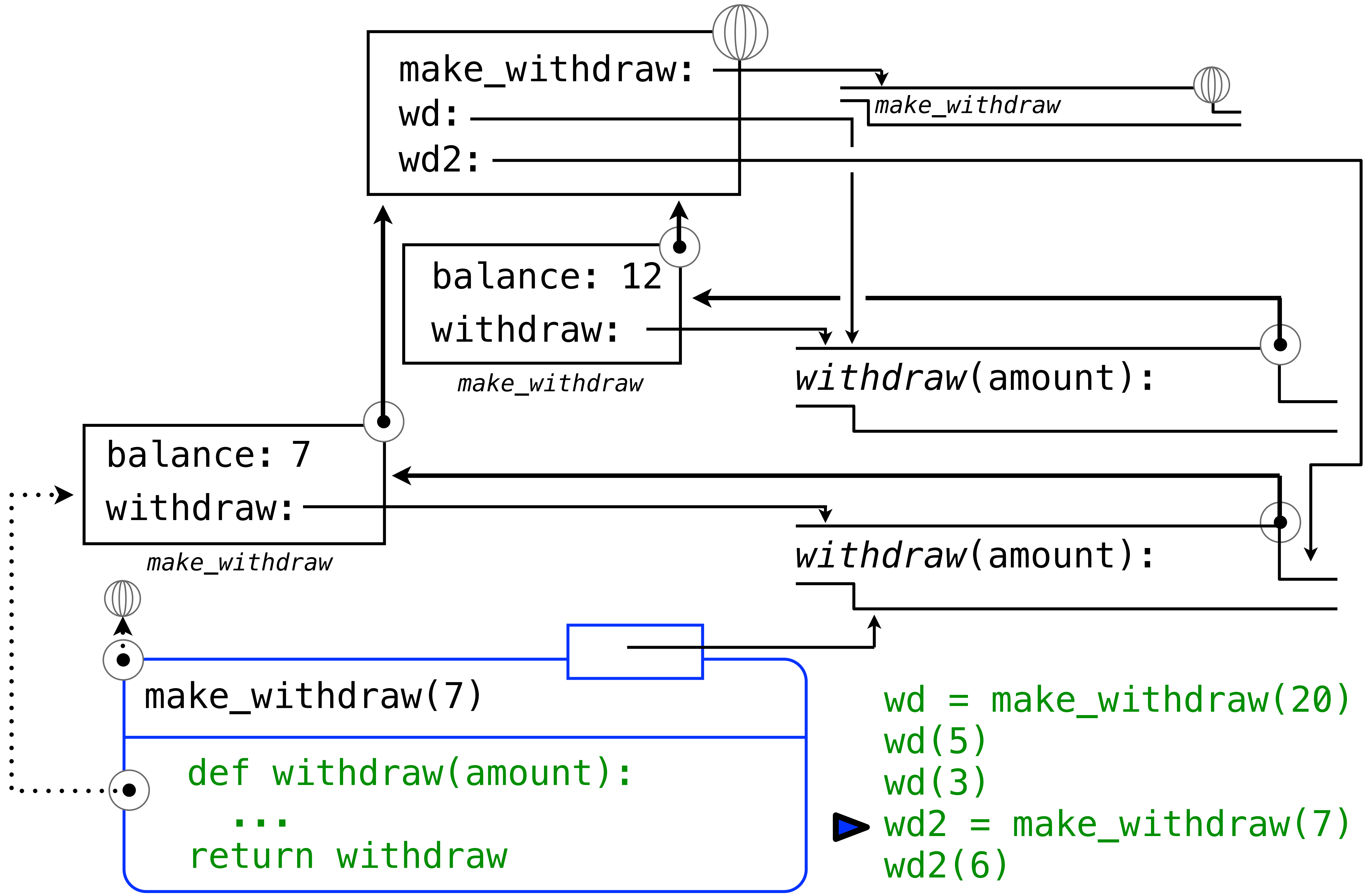
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



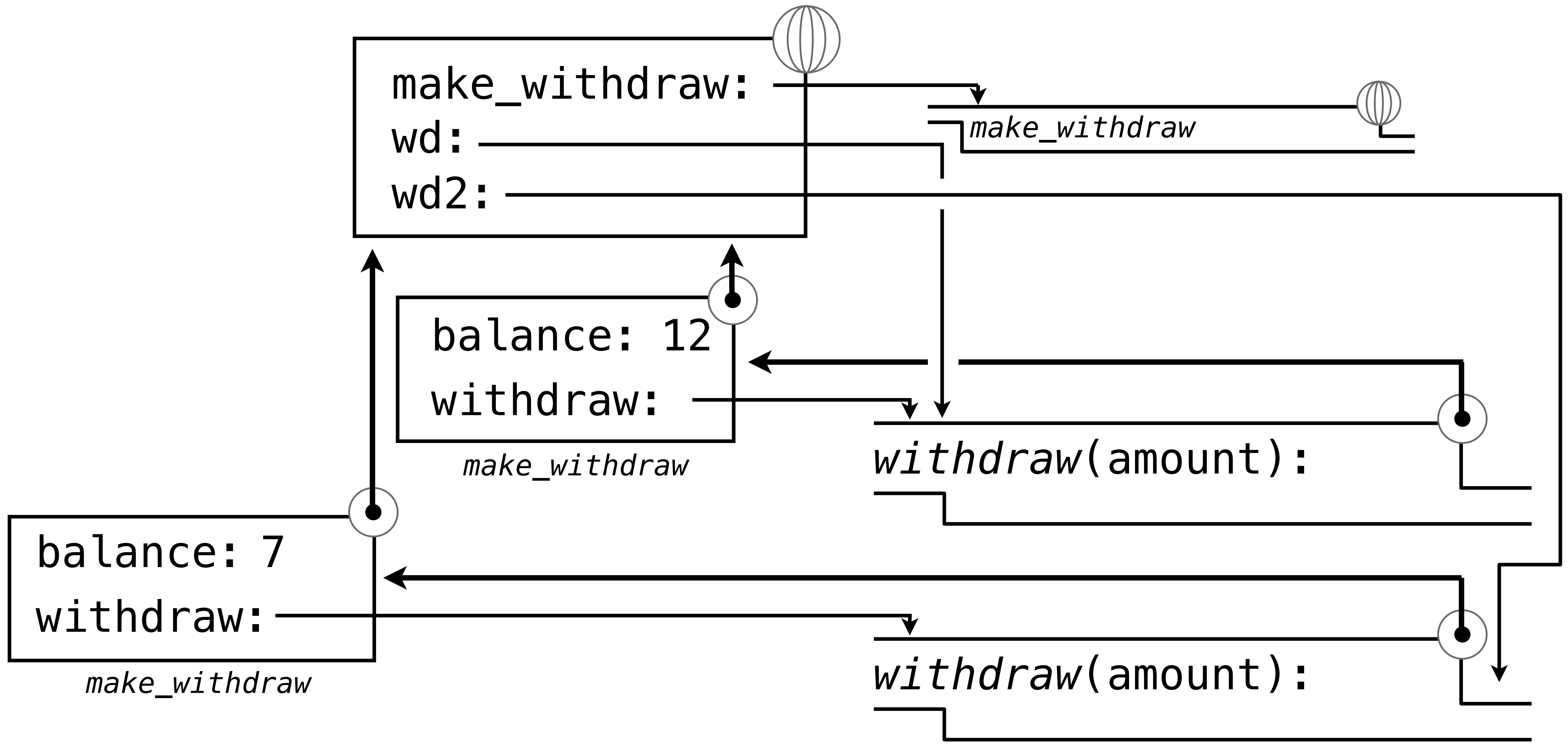
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



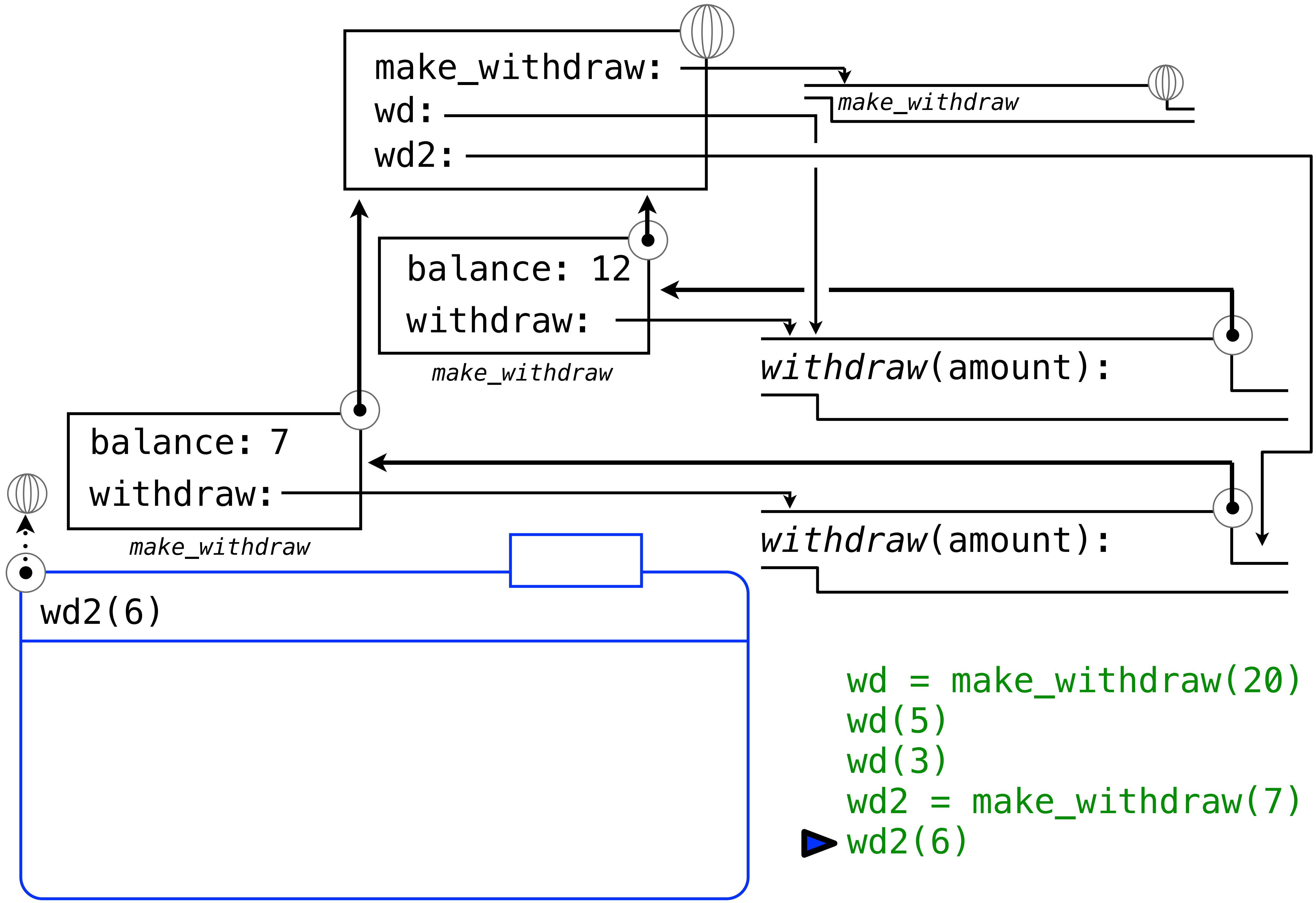
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



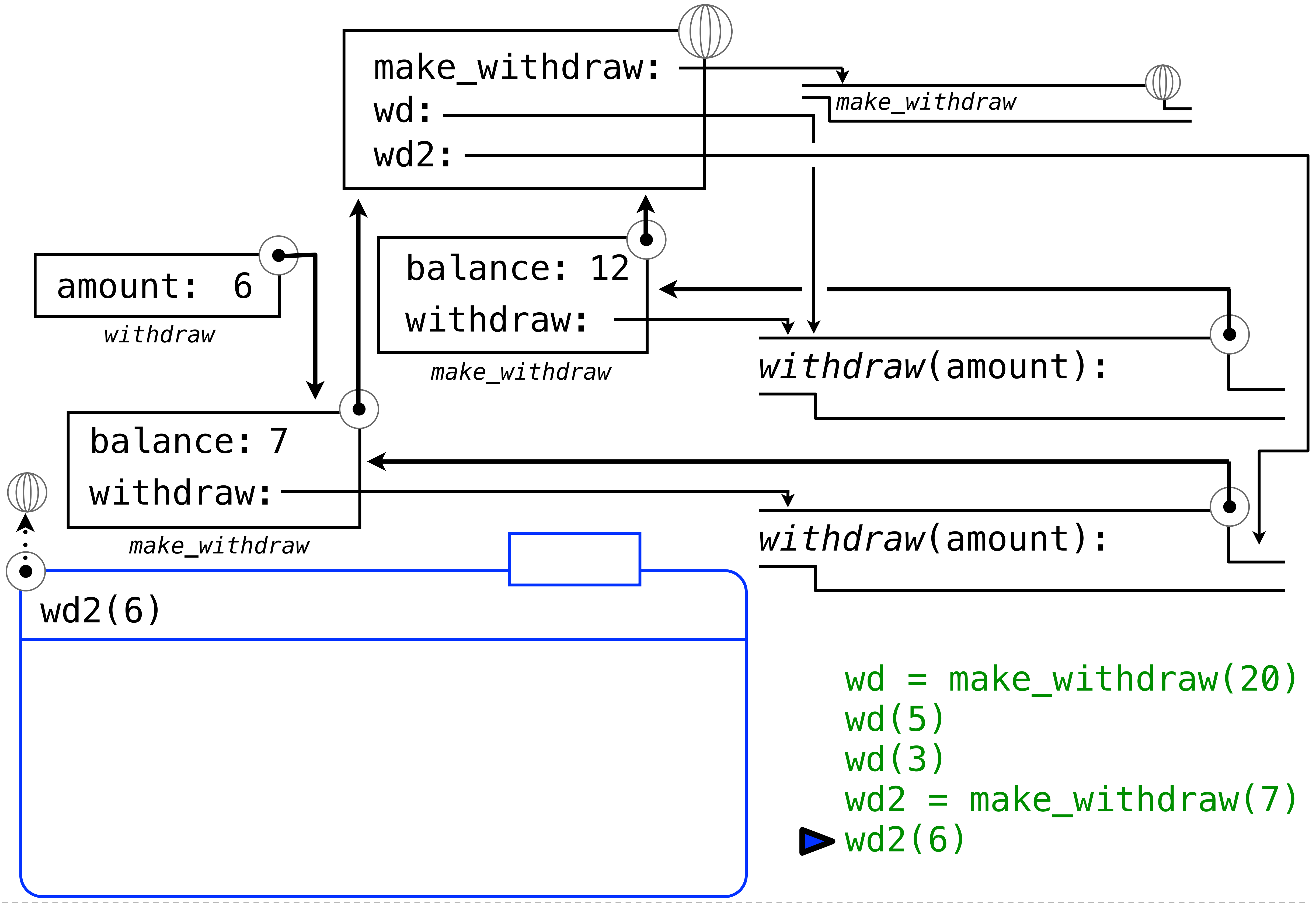
```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

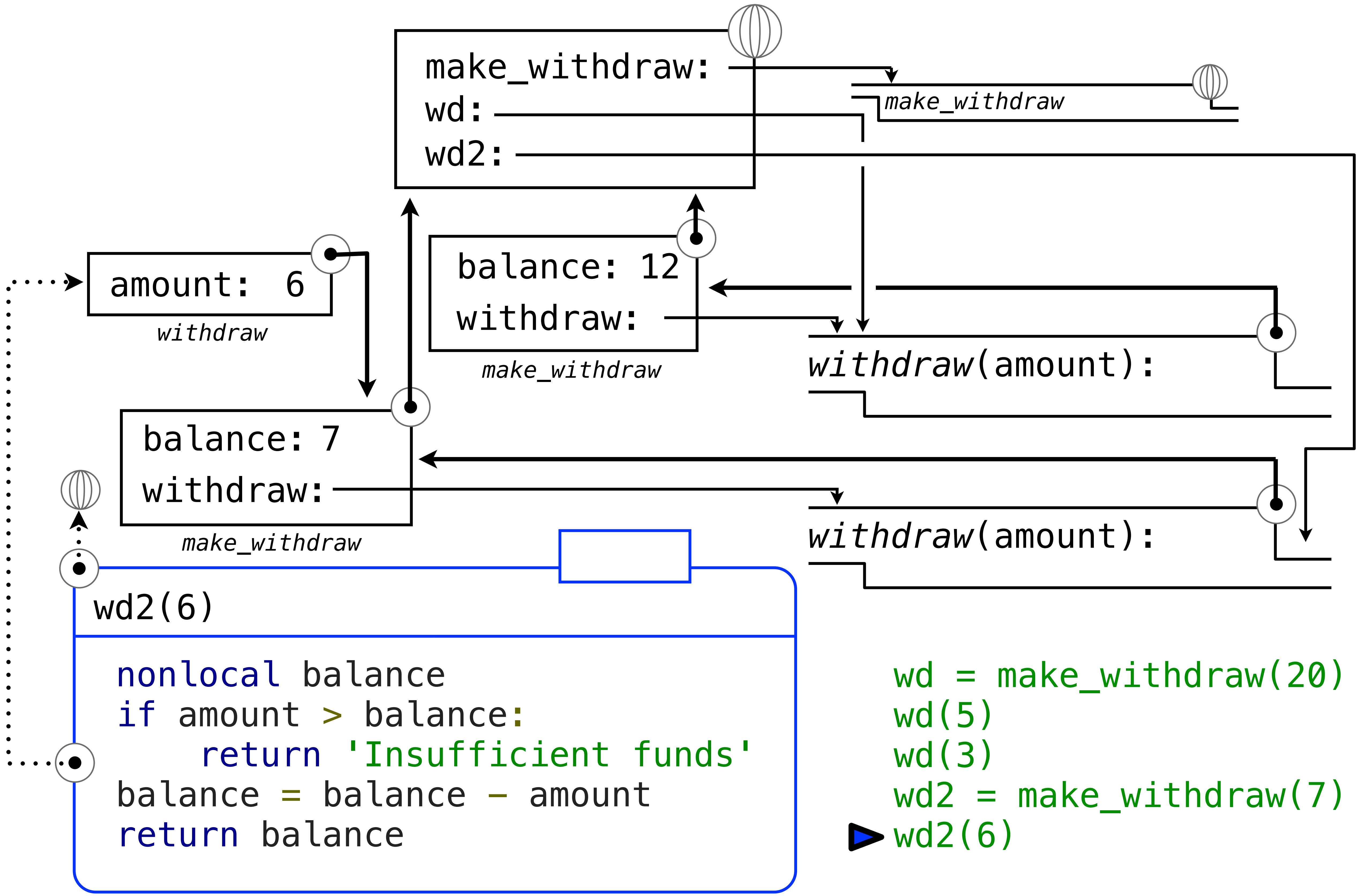
Creating Two Different Withdraw Functions



```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Friday, September 23, 2011

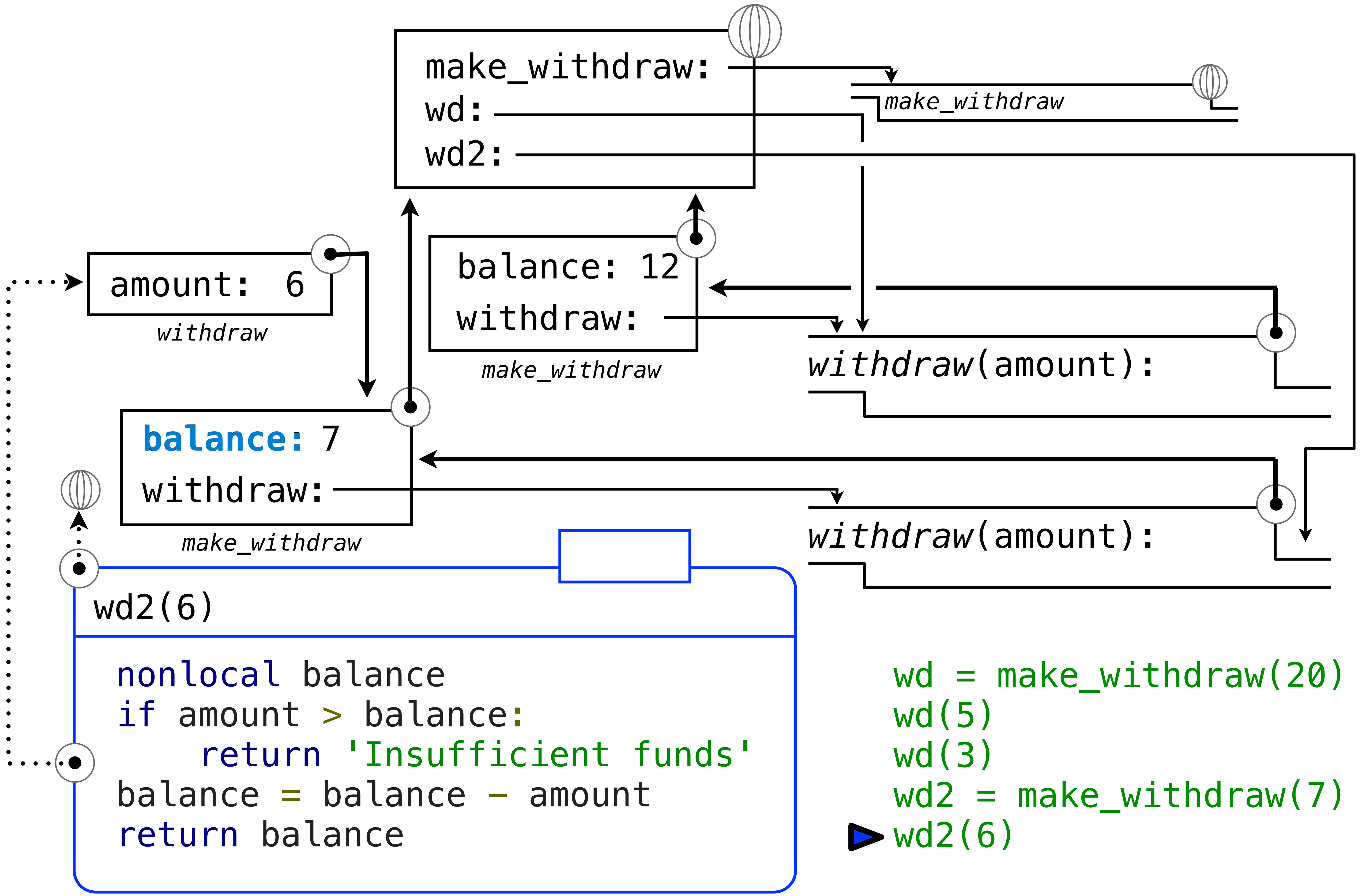
Creating Two Different Withdraw Functions



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

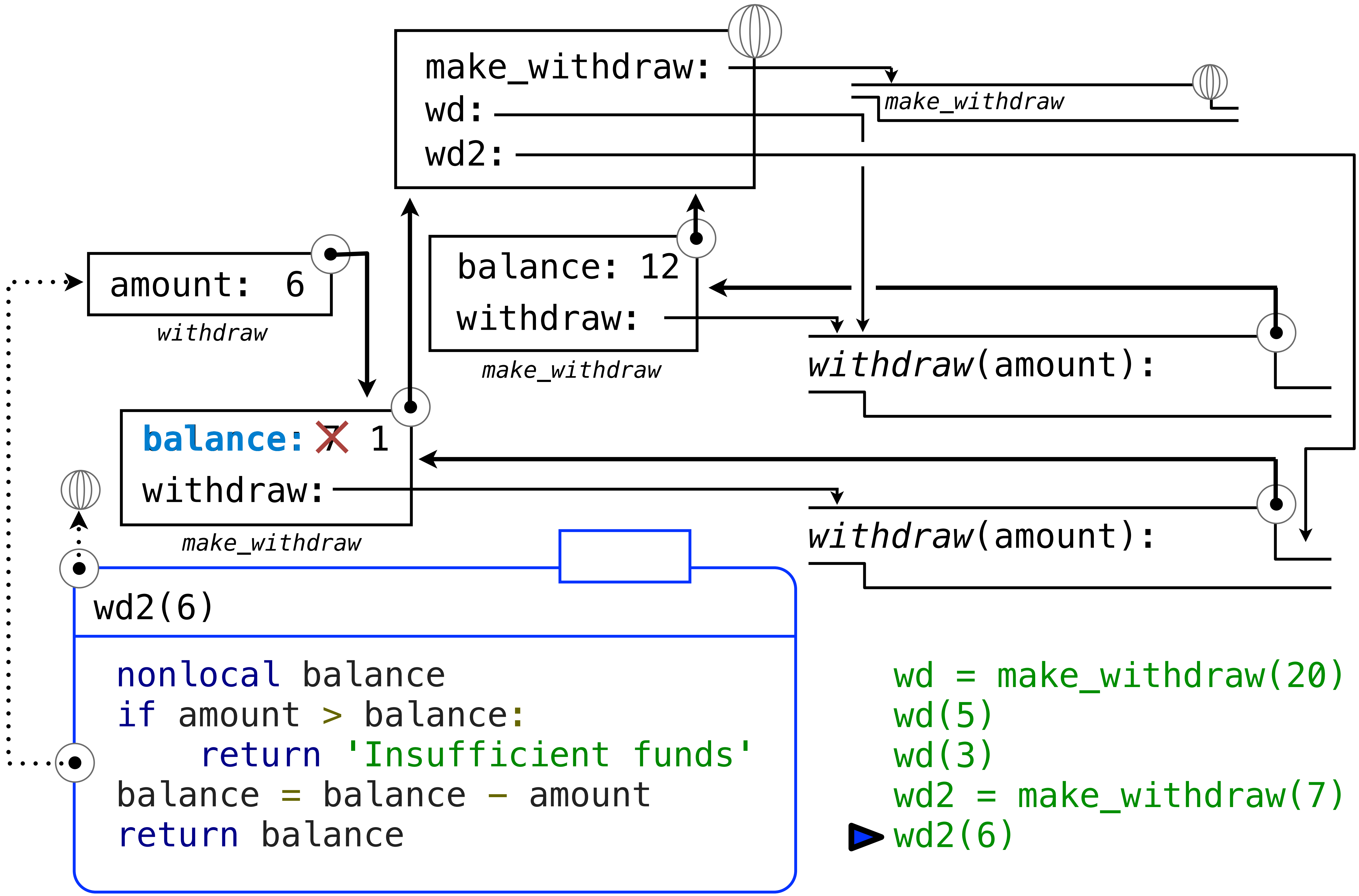
Creating Two Different Withdraw Functions



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

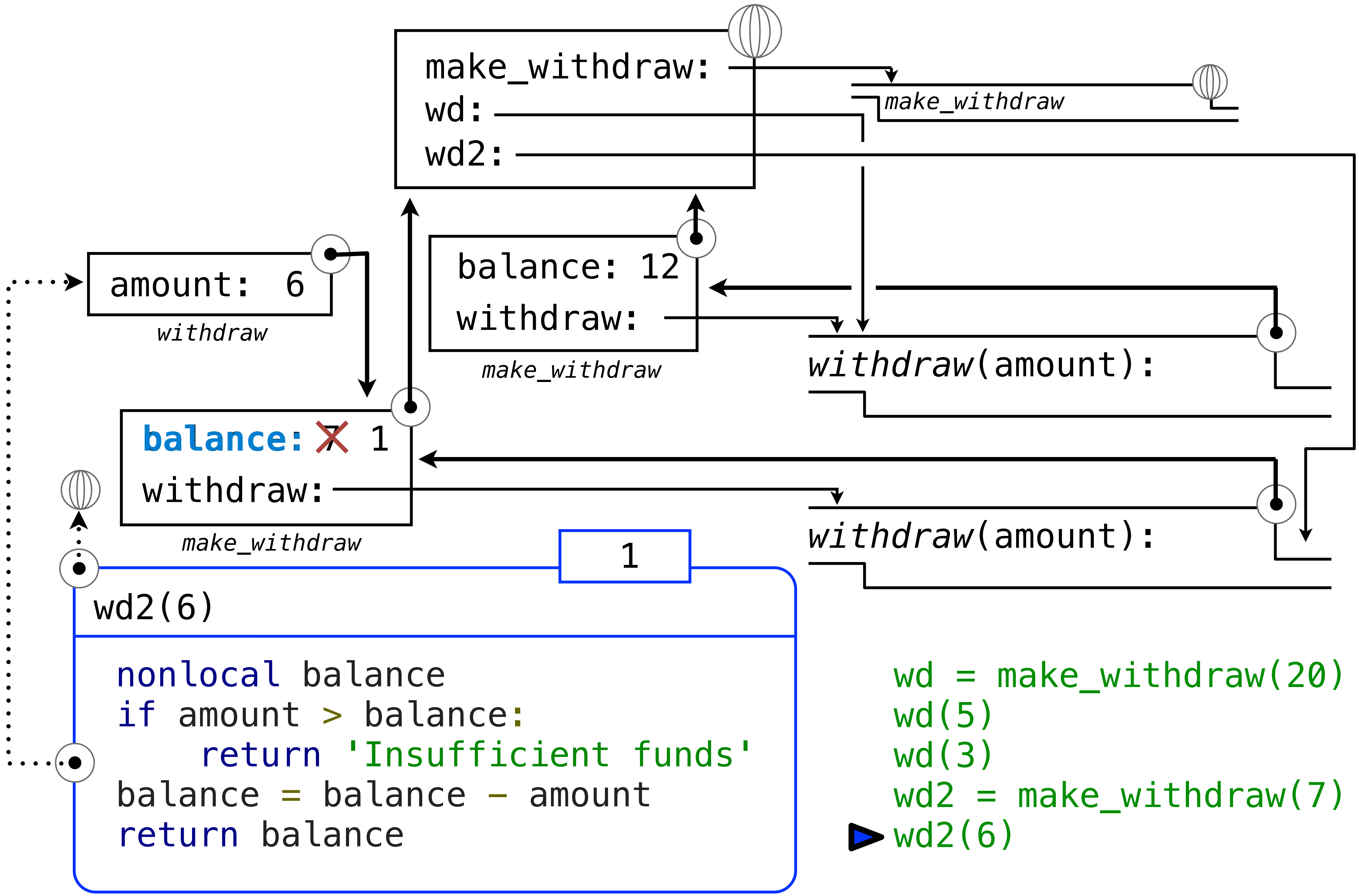
Creating Two Different Withdraw Functions



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

Creating Two Different Withdraw Functions



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(20)
wd(5)
wd(3)
wd2 = make_withdraw(7)
wd2(6)
```

The Benefit of Non-Local Assignment

The Benefit of Non-Local Assignment

- Ability to **maintain some state** that is **local** to a function, but **evolves** over successive calls to that function.

The Benefit of Non-Local Assignment

- Ability to **maintain some state** that is **local** to a function, but **evolves** over successive calls to that function.
- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is **inaccessible to the rest of the program.**

The Benefit of Non-Local Assignment

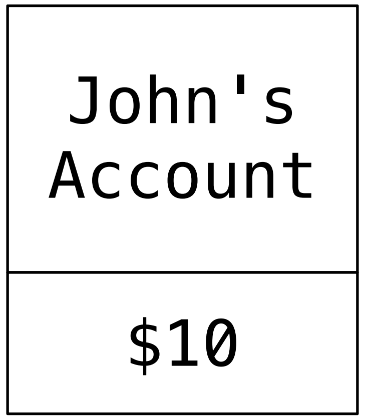
- Ability to **maintain some state** that is **local** to a function, but **evolves** over successive calls to that function.

- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is **inaccessible to the rest of the program.**

- An abstraction of a bank account that **manages its own internal state.**

The Benefit of Non-Local Assignment

- Ability to **maintain some state** that is **local** to a function, but **evolves** over successive calls to that function.
- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is **inaccessible to the rest of the program.**
- An abstraction of a bank account that **manages its own internal state.**

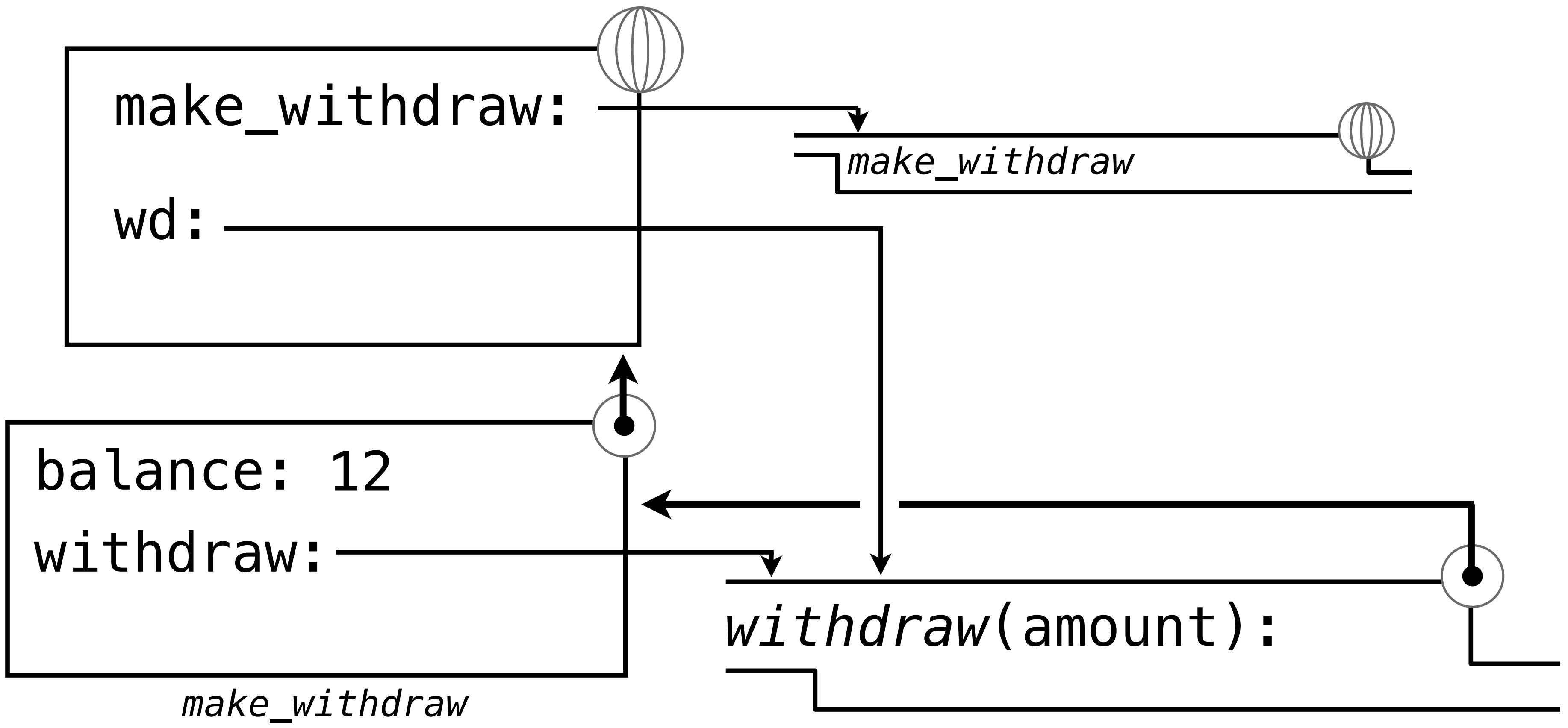


The Benefit of Non-Local Assignment

- Ability to **maintain some state** that is **local** to a function, but **evolves** over successive calls to that function.
- The binding for balance in the first non-local frame of the environment associated with an instance of withdraw is **inaccessible to the rest of the program.**
- An abstraction of a bank account that **manages its own internal state.**

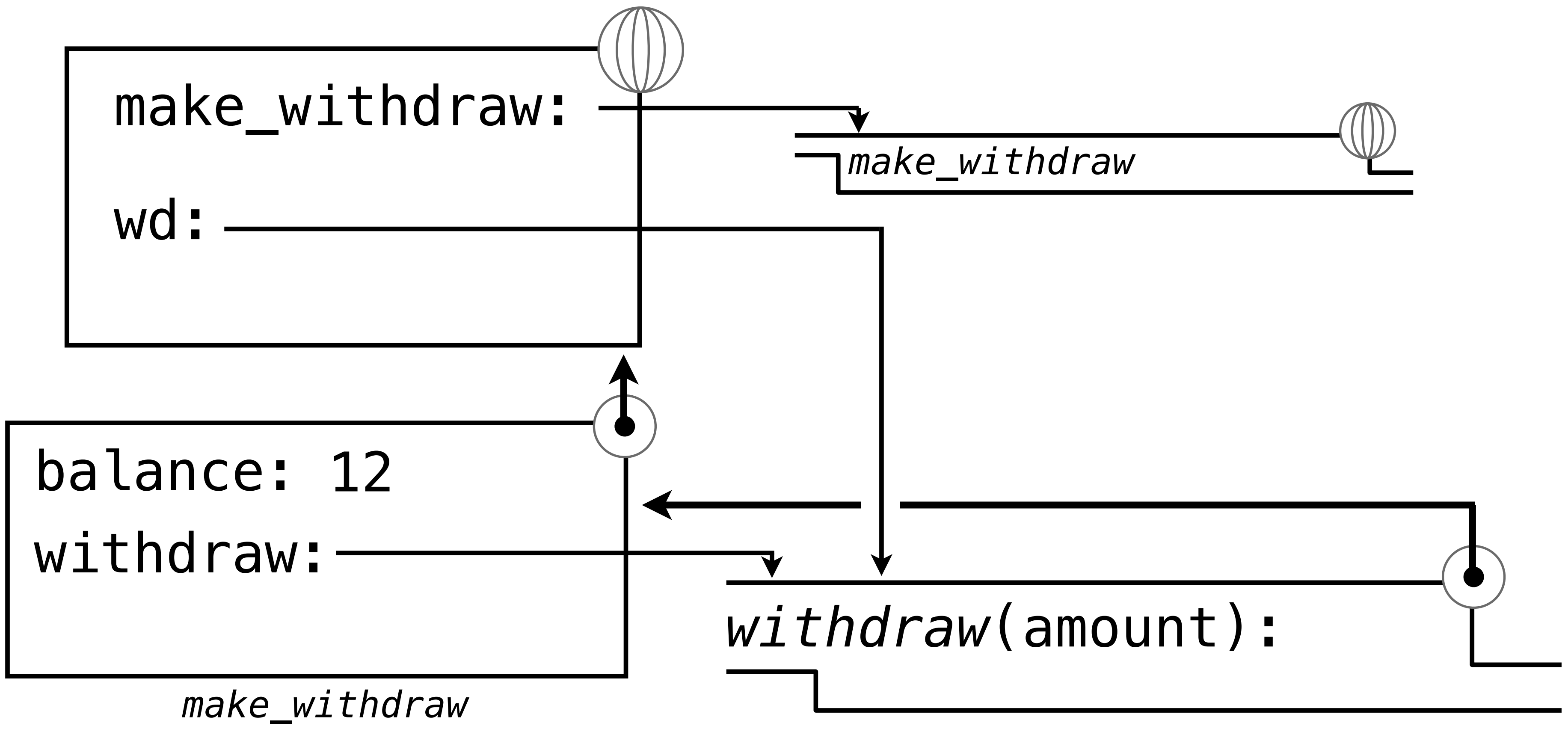
John's Account
$10
Steven's Account
$1,000,000

Multiple References to a Single Withdraw Function



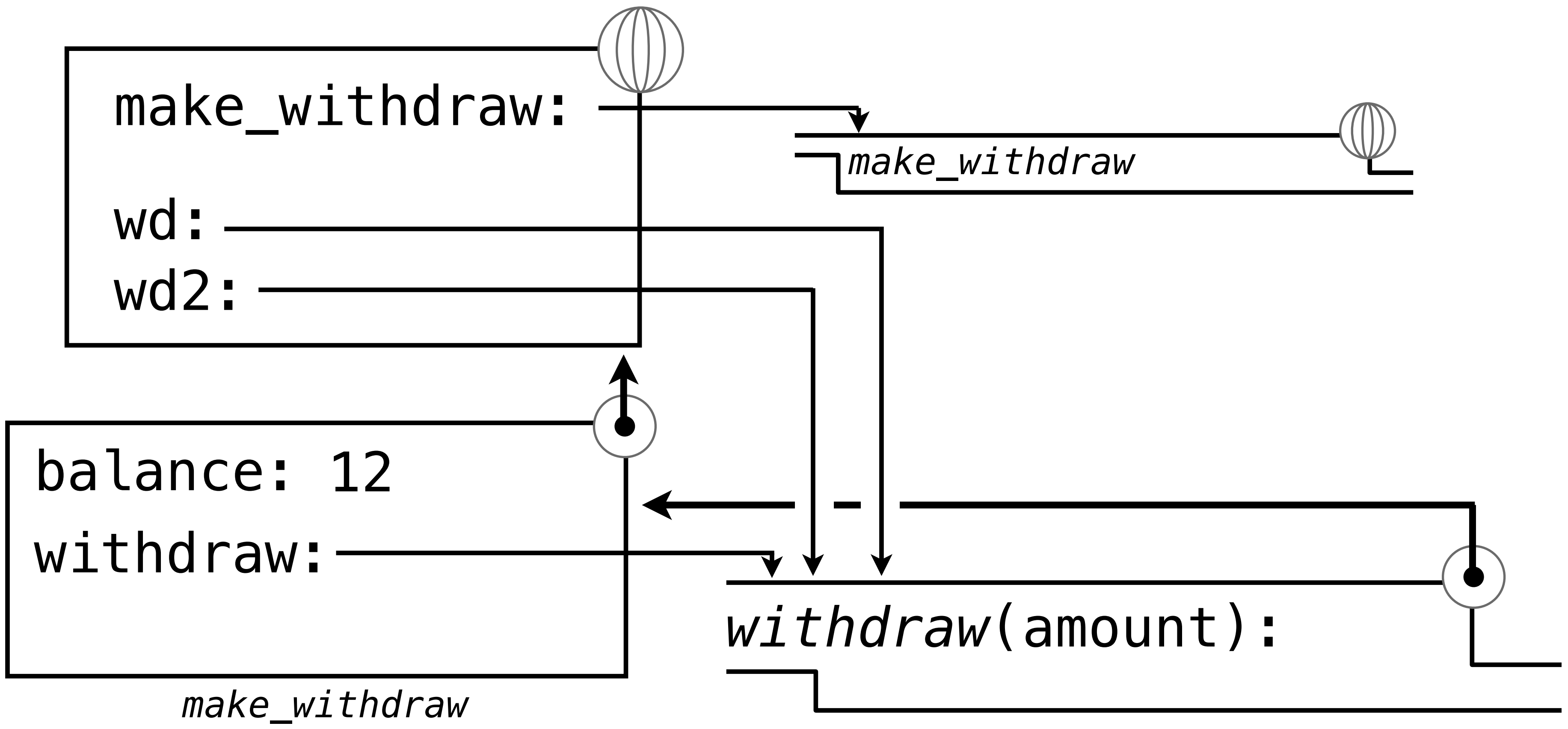
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



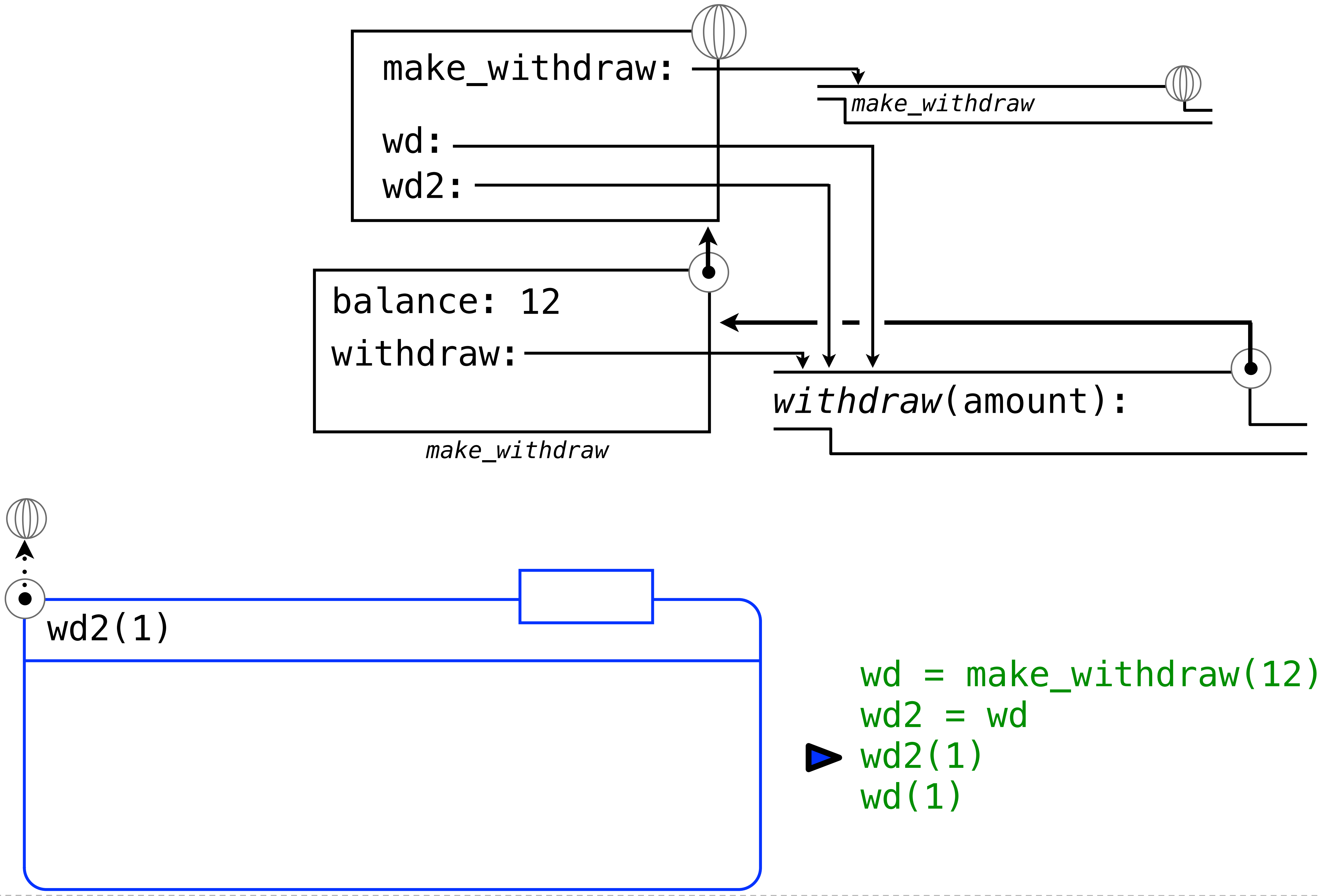
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



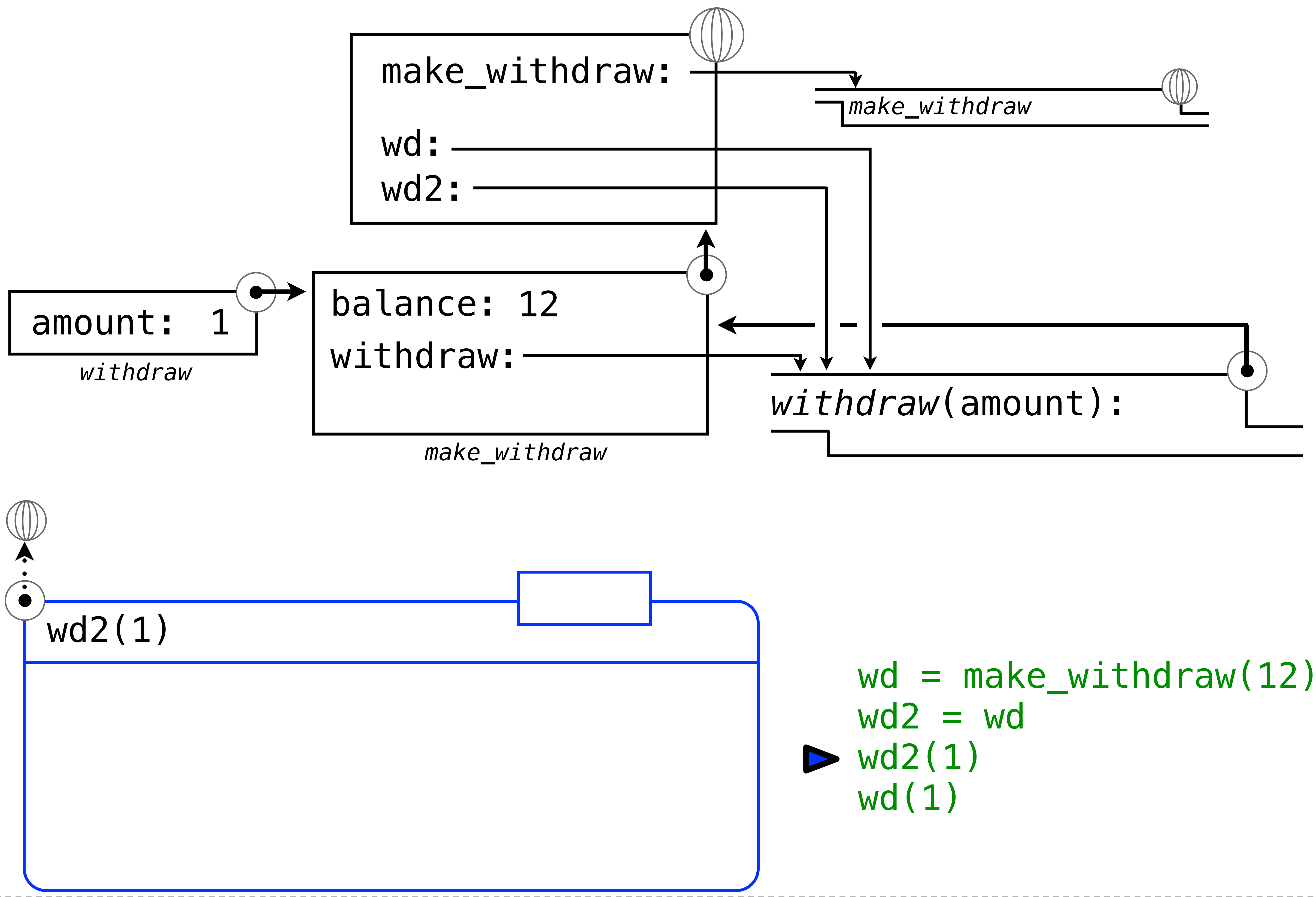
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



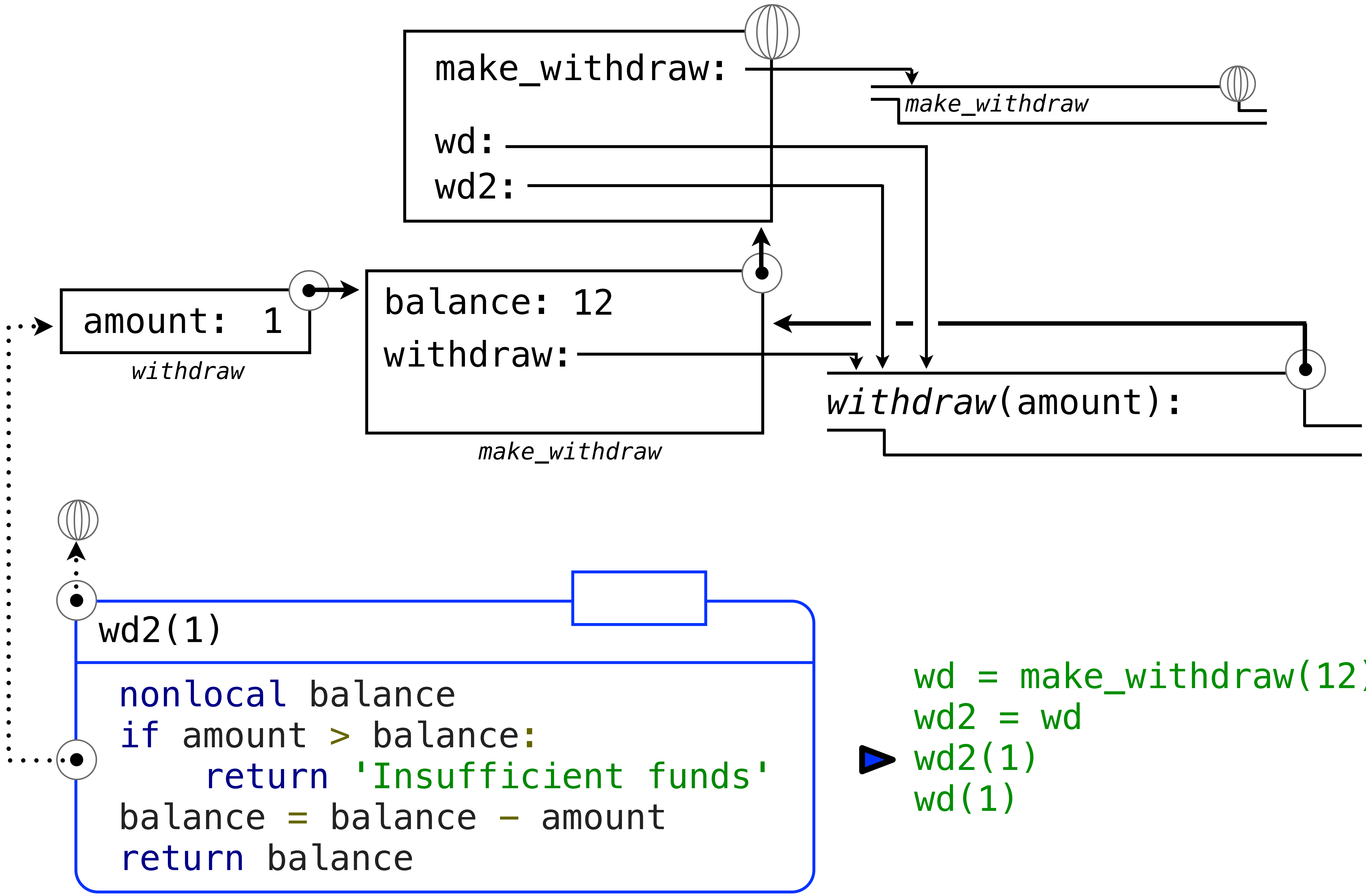
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

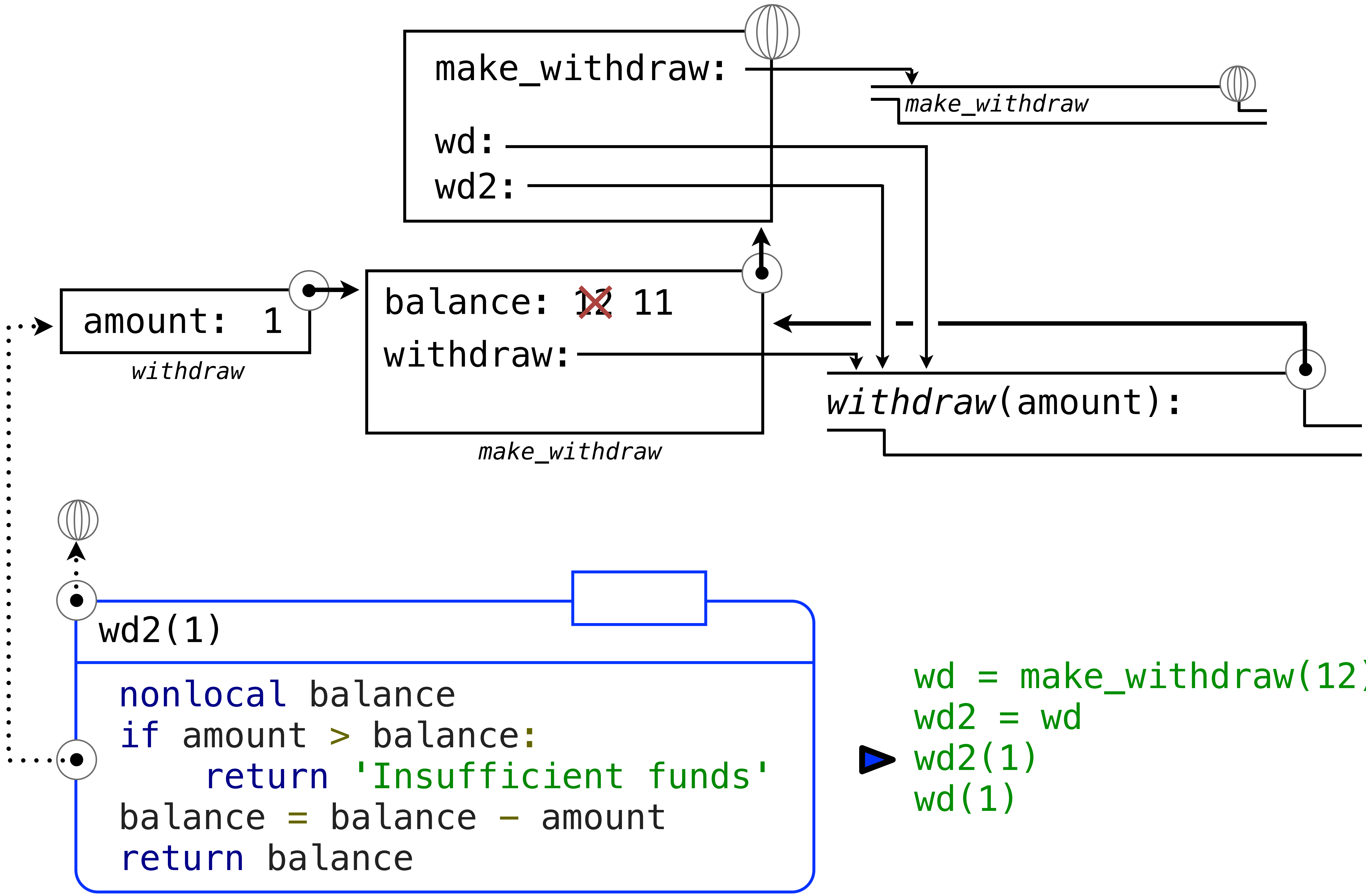
Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

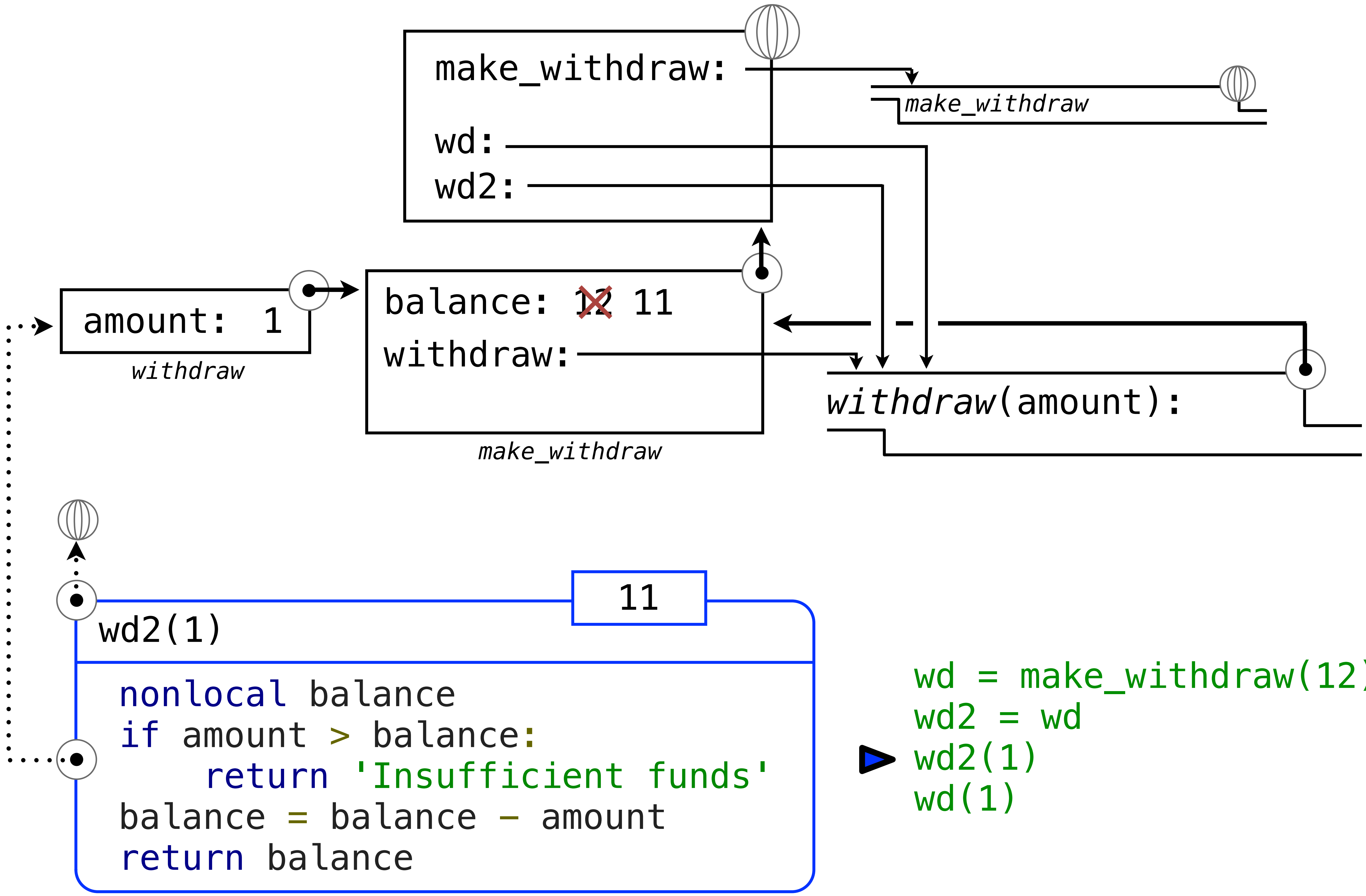
Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

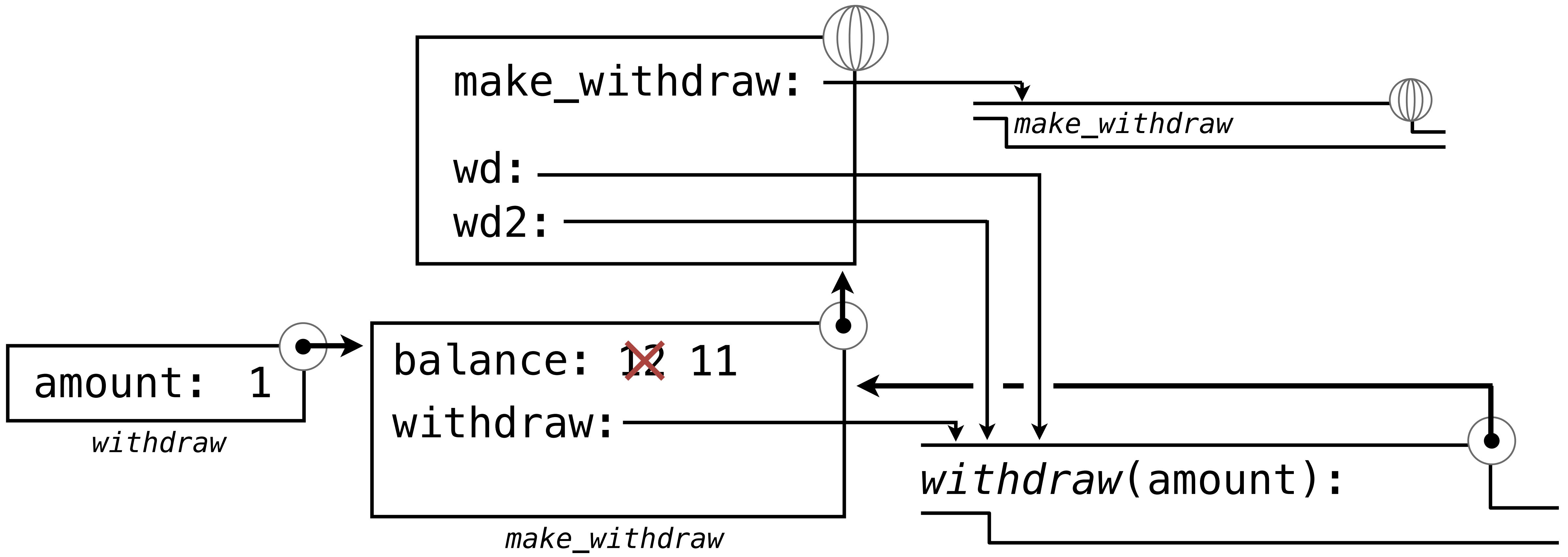
Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

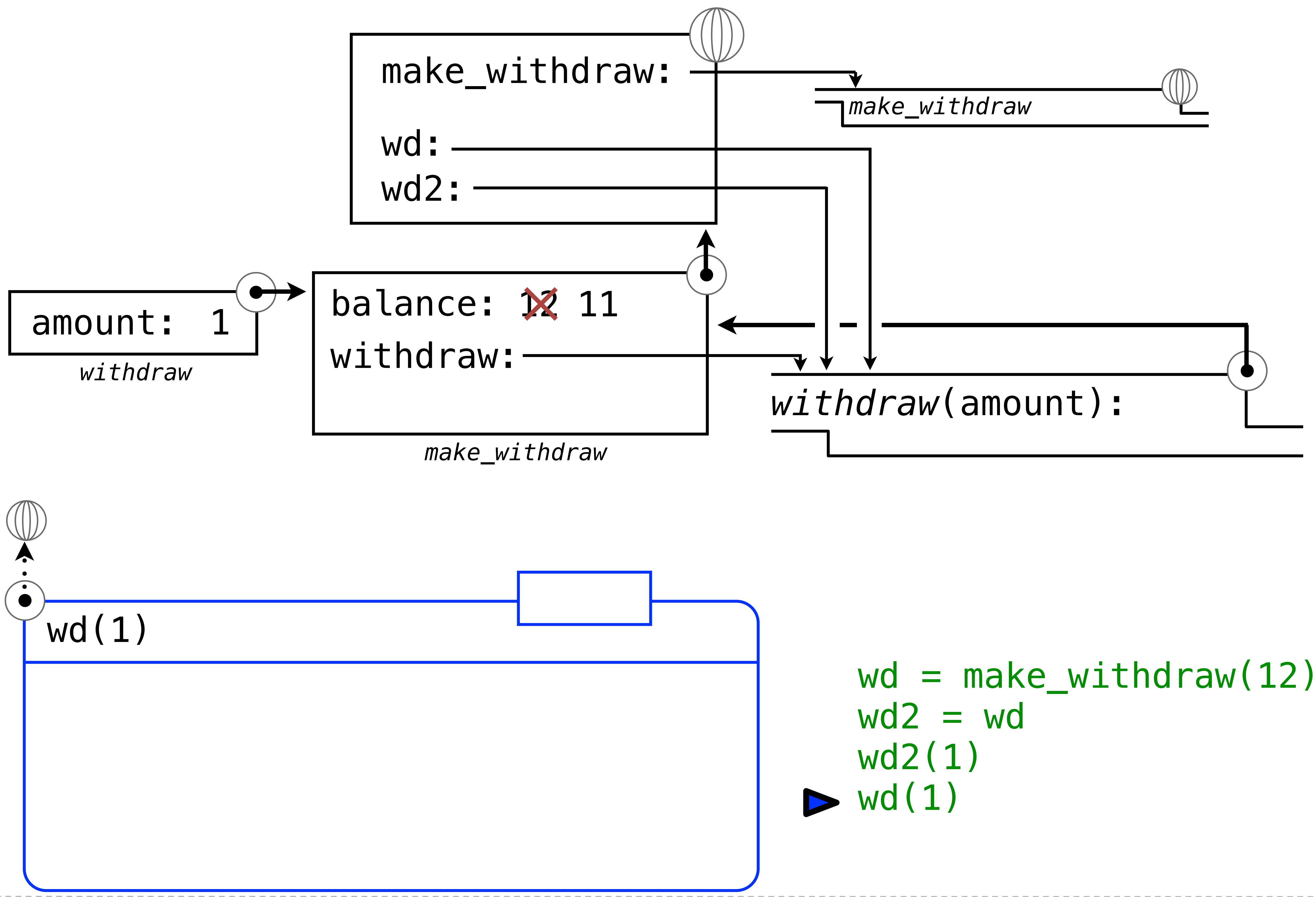
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



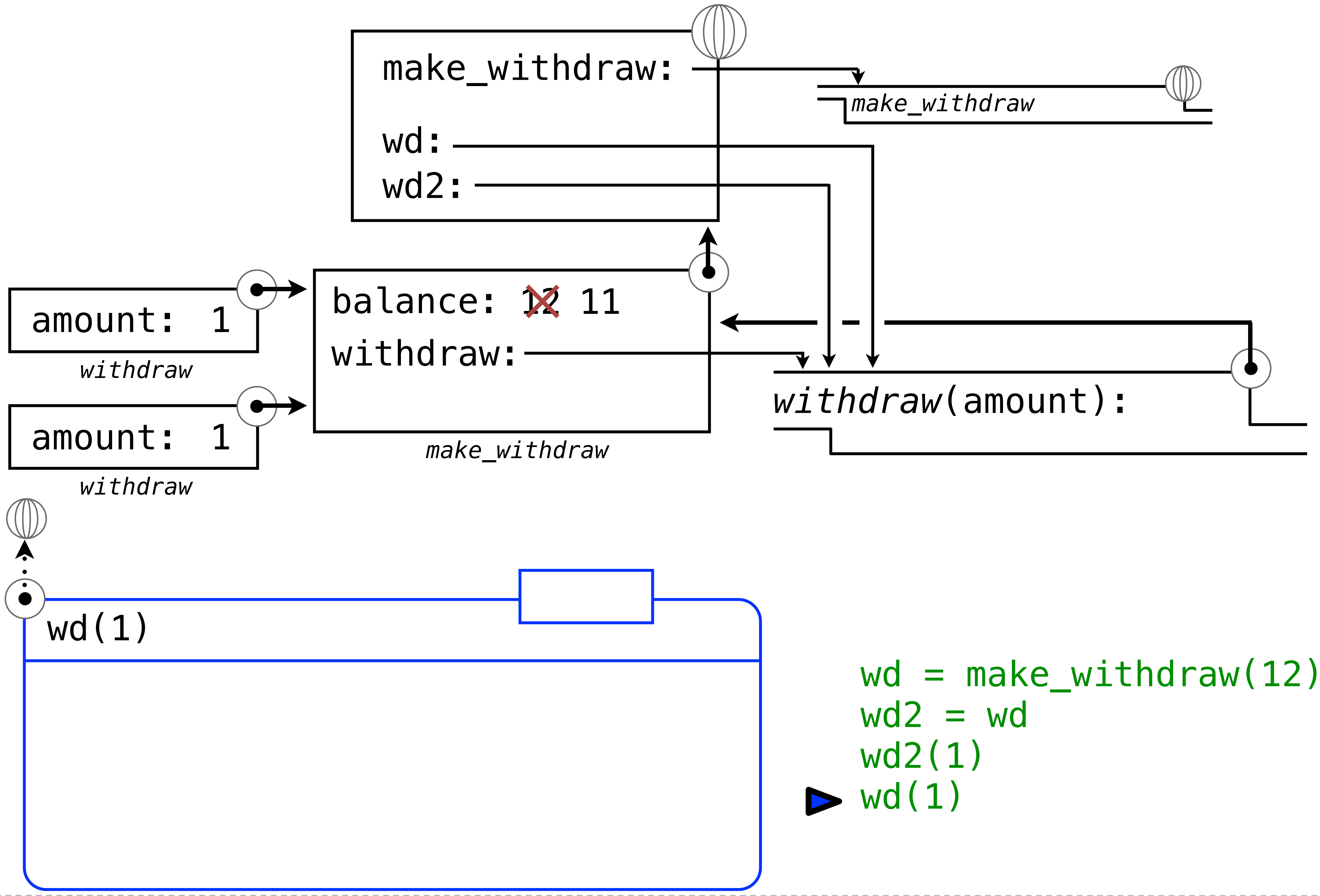
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



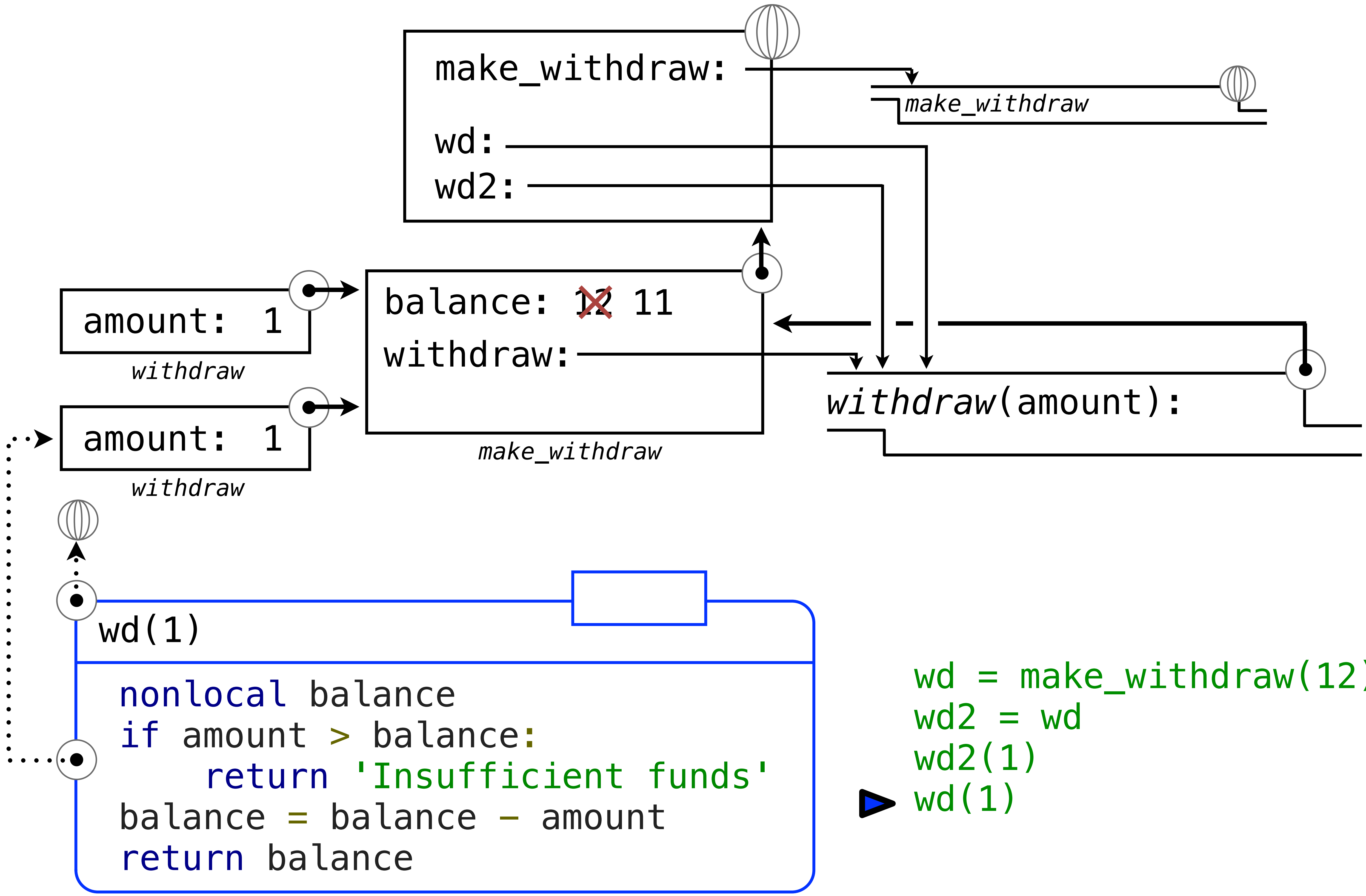
```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

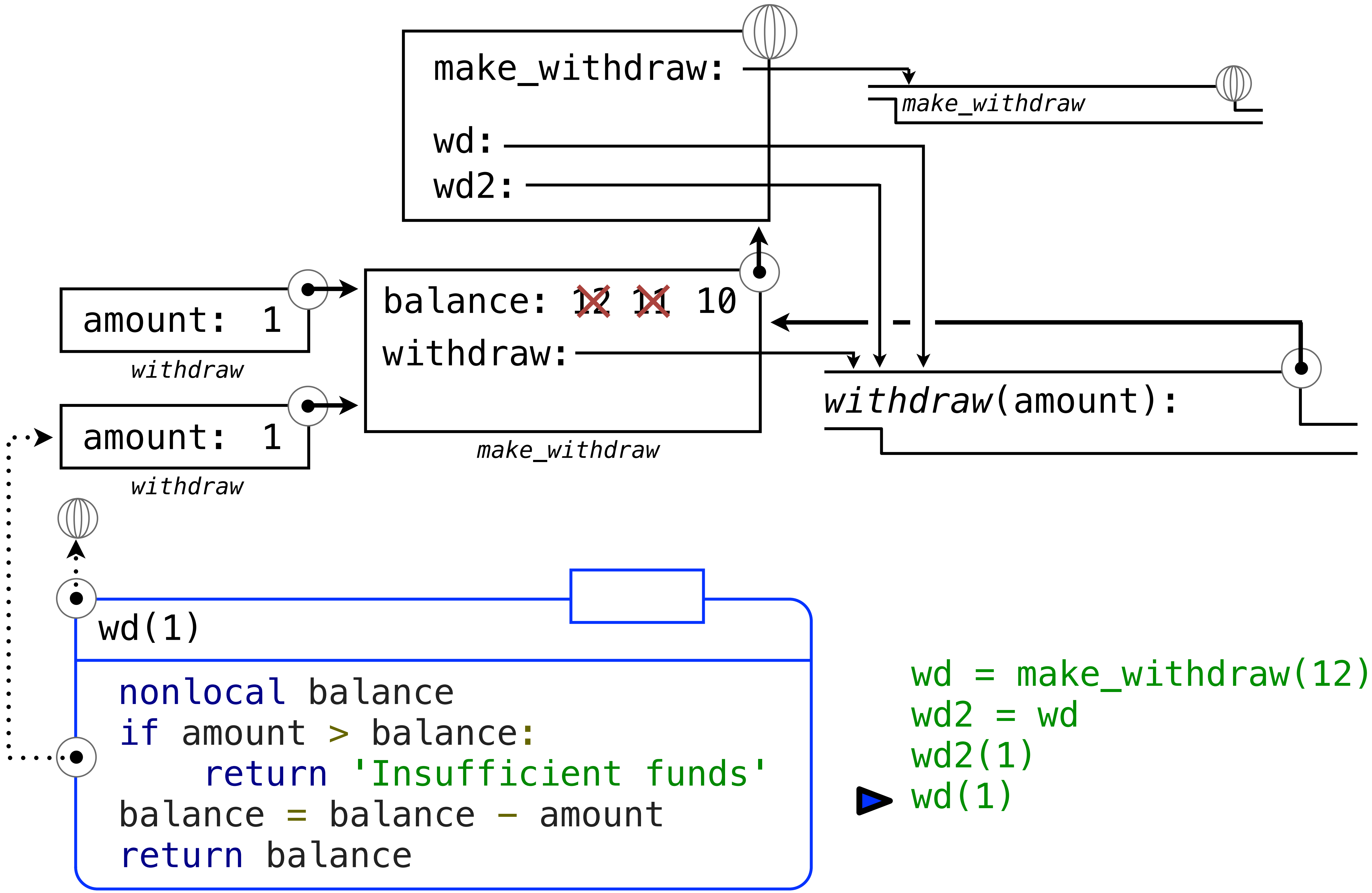
Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

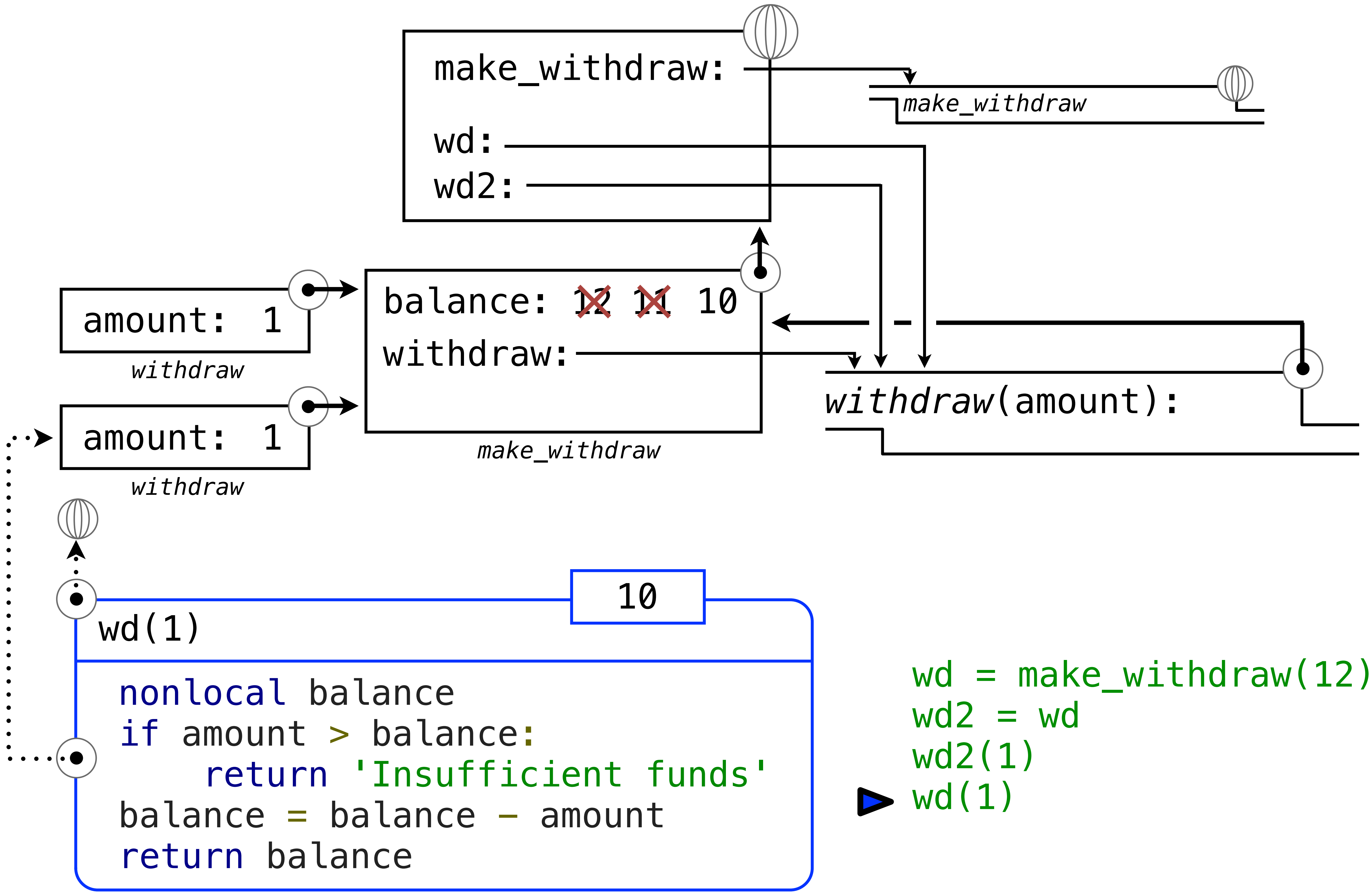
Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Multiple References to a Single Withdraw Function



```
nonlocal balance
if amount > balance:
    return 'Insufficient funds'
balance = balance - amount
return balance
```

```
wd = make_withdraw(12)
wd2 = wd
wd2(1)
wd(1)
```

Sameness and Change

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.

Sameness and Change

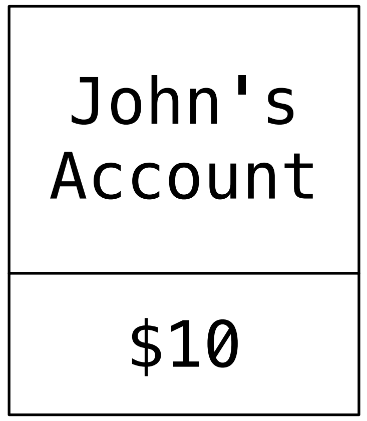
- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.
- A bank account is **still "the same" bank account even if we change the balance** by making a withdrawal.

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.
- A bank account is **still "the same" bank account even if we change the balance** by making a withdrawal.
- Conversely, we could have two bank accounts that happen to have the **same balance, but are different objects.**

Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.
- A bank account is **still "the same" bank account even if we change the balance** by making a withdrawal.
- Conversely, we could have two bank accounts that happen to have the **same balance, but are different objects.**



Sameness and Change

- As long as we **never modify** objects, we can regard a compound object to be precisely the **totality of its pieces.**
- A **rational number** is just its numerator and denominator.
- This view is no longer valid **in the presence of change.**
- Now, a compound data **object has an "identity"** that is something more than the pieces of which it is composed.
- A bank account is **still "the same" bank account even if we change the balance** by making a withdrawal.
- Conversely, we could have two bank accounts that happen to have the **same balance, but are different objects.**

John's Account
$10
Steven's Account
$10

Referential Transparency, Lost

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

```
mul(add(2, mul(4, 6)), add(3, 5))
```

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))
```

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(      26          , add(3, 5))
```

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(      26          , add(3, 5))
```

- Re-binding operations violate the condition of referential transparency because they do more than return a value; **they change the environment.**

Referential Transparency, Lost

- An expression is **referentially transparent** if its value does not change when we substitute one of its subexpression with the value of that subexpression.

```
mul(add(2, mul(4, 6)), add(3, 5))

mul(add(2,    24    ), add(3, 5))

mul(      26          , add(3, 5))
```

- Re-binding operations violate the condition of referential transparency because they do more than return a value; **they change the environment.**
- Two separately defined functions are not the same, because **changes to one may not be reflected in the other.**