

Attribute assignment statements can change first and rest attributes of a Link

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

Attribute assignment statements can change first and rest attributes of a Link

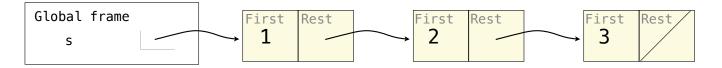
The rest of a linked list can contain the linked list as a sub-list

>>> s = Link(1, Link(2, Link(3)))

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

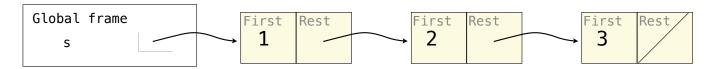
>>> s = Link(1, Link(2, Link(3)))



Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

>>> s = Link(1, Link(2, Link(3)))



Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
```

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.first
```

Attribute assignment statements can change first and rest attributes of a Link

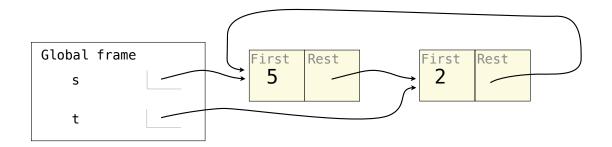
The rest of a linked list can contain the linked list as a sub-list

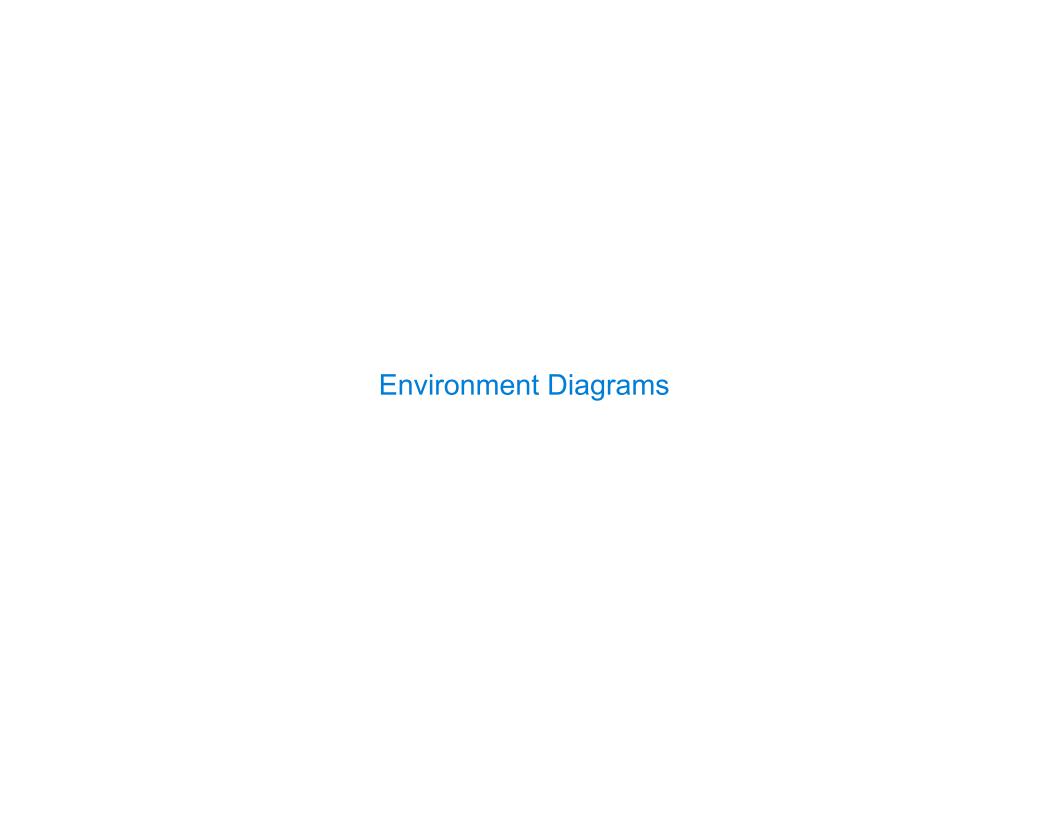
```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.first
2
```

Attribute assignment statements can change first and rest attributes of a Link

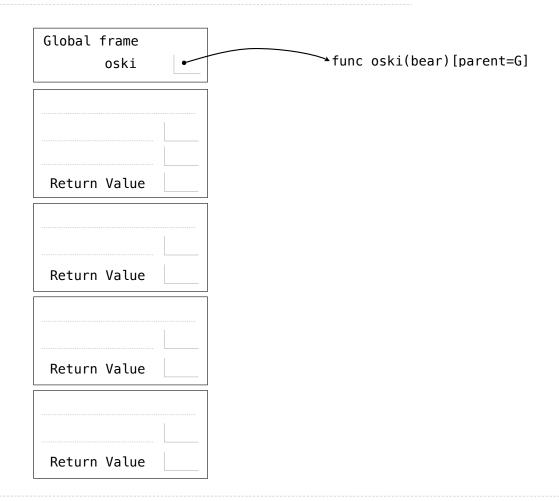
The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.first
2
```

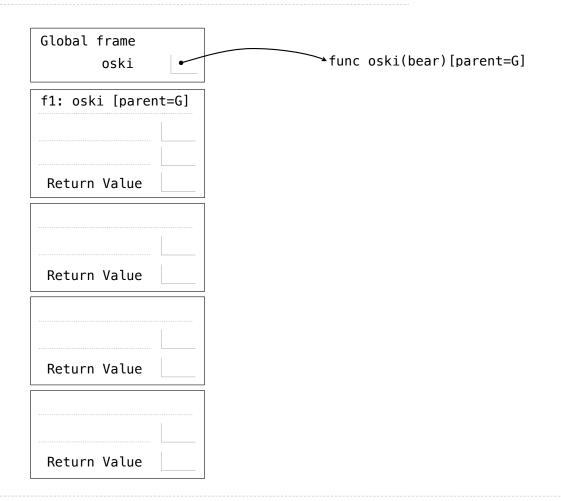




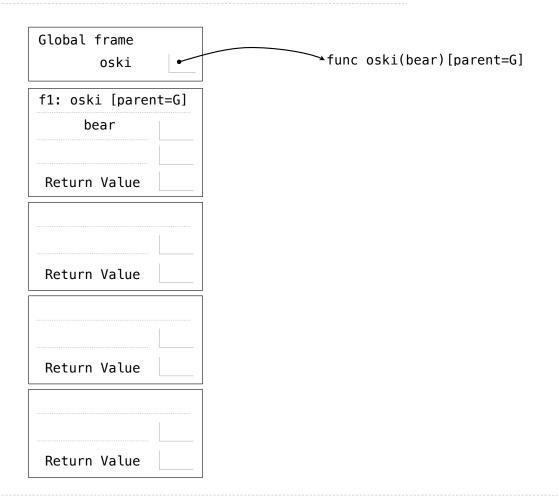
```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
        return cal(2)
oski(abs)
```



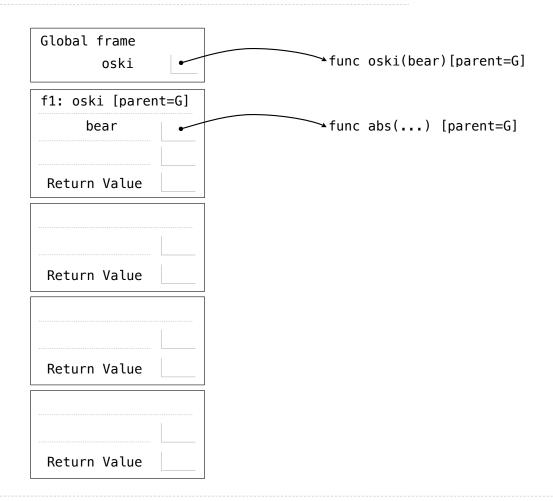
```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
        return cal(2)
oski(abs)
```



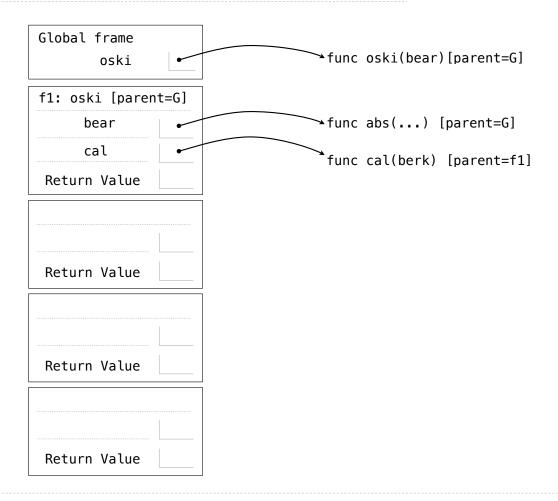
```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
        return cal(2)
oski(abs)
```



```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
        return cal(2)
oski(abs)
```



```
def oski(bear):
    def cal(berk):
        nonlocal bear
        if bear(berk) == 0:
            return [berk+1, berk-1]
        bear = lambda ley: berk-ley
        return [berk, cal(berk)]
        return cal(2)
oski(abs)
```



```
Global frame
def oski(bear):
                                                                               →func oski(bear)[parent=G]
                                                       oski
    def cal(berk):
                                                f1: oski [parent=G]
         nonlocal bear
                                                                               →func abs(...) [parent=G]
                                                     bear
         if bear(berk) == 0:
                                                     cal
                                                                                func cal(berk) [parent=f1]
                                                 Return Value
              return [berk+1, berk-1]
         bear = lambda ley: berk-ley :
         return [berk, cal(berk)]
                                                 Return Value
    return cal(2)
oski(abs)
                                                 Return Value
                                                 Return Value
```

```
Global frame
def oski(bear):
                                                                               →func oski(bear)[parent=G]
                                                       oski
    def cal(berk):
                                                f1: oski [parent=G]
         nonlocal bear
                                                                              →func abs(...) [parent=G]
                                                     bear
         if bear(berk) == 0:
                                                     cal
                                                                                func cal(berk) [parent=f1]
                                                Return Value
              return [berk+1, berk-1]
                                                f2: cal
         bear = lambda ley: berk-ley
         return [berk, cal(berk)]
                                                Return Value
    return cal(2)
oski(abs)
                                                Return Value
                                                Return Value
```

```
Global frame
def oski(bear):
                                                                               →func oski(bear)[parent=G]
                                                       oski
    def cal(berk):
                                                f1: oski [parent=G]
         nonlocal bear
                                                                               →func abs(...) [parent=G]
                                                     bear
         if bear(berk) == 0:
                                                     cal
                                                                                func cal(berk) [parent=f1]
                                                 Return Value
              return [berk+1, berk-1]
                                                f2: cal [parent=f1]
         bear = lambda ley: berk-ley
         return [berk, cal(berk)]
                                                 Return Value
    return cal(2)
oski(abs)
                                                 Return Value
                                                 Return Value
```

```
Global frame
def oski(bear):
                                                                                →func oski(bear)[parent=G]
                                                       oski
    def cal(berk):
                                                f1: oski [parent=G]
         nonlocal bear
                                                                               →func abs(...) [parent=G]
                                                     bear
         if bear(berk) == 0:
                                                     cal
                                                                                 func cal(berk) [parent=f1]
                                                 Return Value
              return [berk+1, berk-1]
                                                f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                         berk
         return [berk, cal(berk)]
                                                 Return Value
    return cal(2)
oski(abs)
                                                 Return Value
                                                 Return Value
```

```
Global frame
def oski(bear):
                                                                                  →func oski(bear)[parent=G]
                                                         oski
    def cal(berk):
                                                                                  func \lambda(ley) [parent=f2]
                                                 f1: oski [parent=G]
         nonlocal bear
                                                                                 →func abs(...) [parent=G]
                                                       bear
         if bear(berk) == 0:
                                                      cal
                                                                                  func cal(berk) [parent=f1]
                                                  Return Value
              return [berk+1, berk-1]
                                                 f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                          berk
         return [berk, cal(berk)]
                                                  Return Value
    return cal(2)
oski(abs)
                                                  Return Value
                                                  Return Value
```

```
Global frame
def oski(bear):
                                                                                →func oski(bear)[parent=G]
                                                        oski
    def cal(berk):
                                                 f1: oski [parent=G]
                                                                                →func λ(ley) [parent=f2]
         nonlocal bear
                                                                                →func abs(...) [parent=G]
                                                      bear
         if bear(berk) == 0:
                                                      cal
                                                                                 func cal(berk) [parent=f1]
                                                 Return Value
              return [berk+1, berk-1]
                                                 f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                         berk
         return [berk, cal(berk)]
                                                 Return Value
    return cal(2)
oski(abs)
                                                 Return Value
                                                 Return Value
```

```
Global frame
def oski(bear):
                                                                                    →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                    \rightarrow func \lambda(ley) [parent=f2]
                                                   f1: oski [parent=G]
          nonlocal bear
                                                                                    →func abs(...) [parent=G]
                                                        bear
          if bear(berk) == 0:
                                                        cal
                                                                                     func cal(berk) [parent=f1]
                                                    Return Value
               return [berk+1, berk-1]
                                                   f2: cal [parent=f1]
          bear = lambda ley: berk-ley
                                                            berk
          return [berk, cal(berk)]
                                                    Return Value
     return cal(2)
oski(abs)
                                                    Return Value
                                                    Return Value
```

```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
    def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
         nonlocal bear
                                                        bear
         if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                            berk
         return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

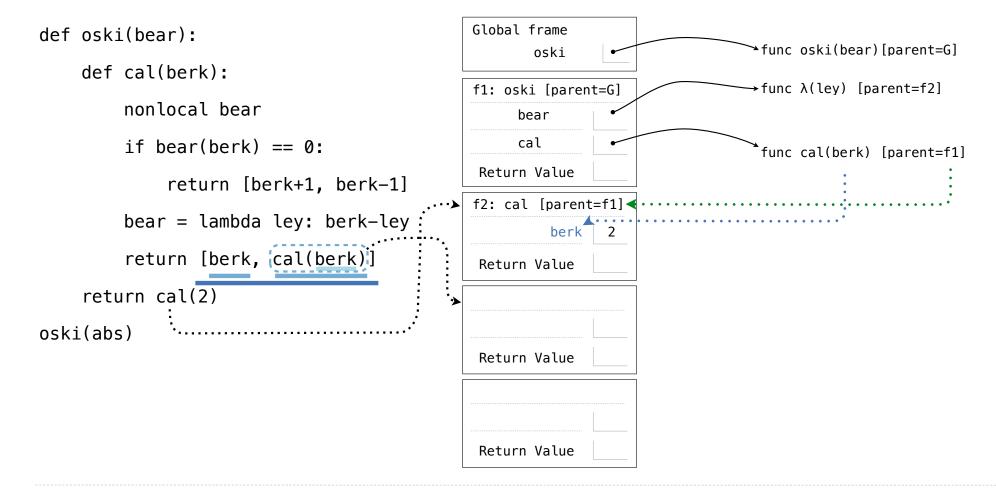
```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
         nonlocal bear
                                                        bear
         if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                            berk
         return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

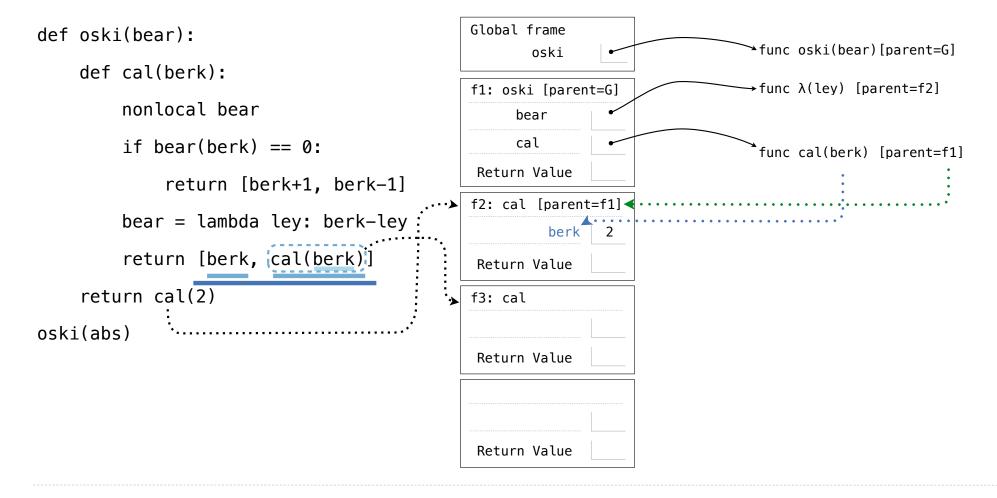
```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
    def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
         nonlocal bear
                                                        bear
         if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                            berk
         return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
    def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
         nonlocal bear
                                                        bear
         if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                            berk
          return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
         nonlocal bear
                                                        bear
         if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
         bear = lambda ley: berk-ley
                                                            berk
          return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

```
Global frame
def oski(bear):
                                                                                   →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                   \rightarrow func \lambda(ley) [parent=f2]
                                                  f1: oski [parent=G]
          nonlocal bear
                                                        bear
          if bear(berk) == 0:
                                                        cal
                                                                                    func cal(berk) [parent=f1]
                                                   Return Value
               return [berk+1, berk-1]
                                                  f2: cal [parent=f1]
          bear = lambda ley: berk-ley
                                                            berk
          return [berk, cal(berk)]
                                                   Return Value
     return cal(2)
oski(abs)
                                                   Return Value
                                                   Return Value
```

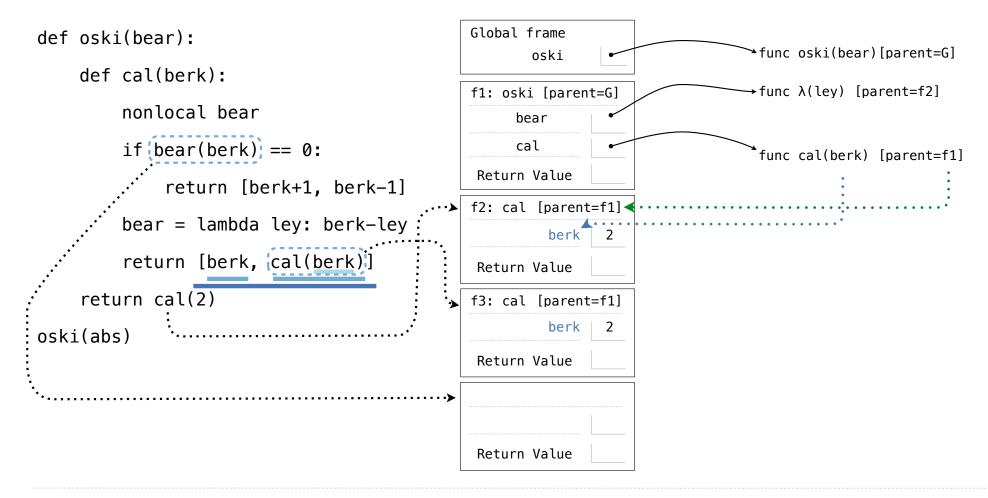


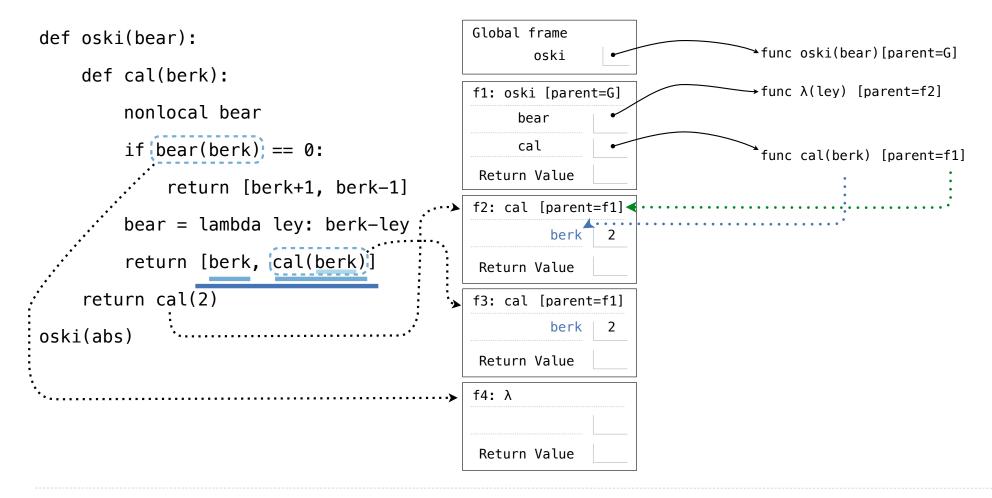


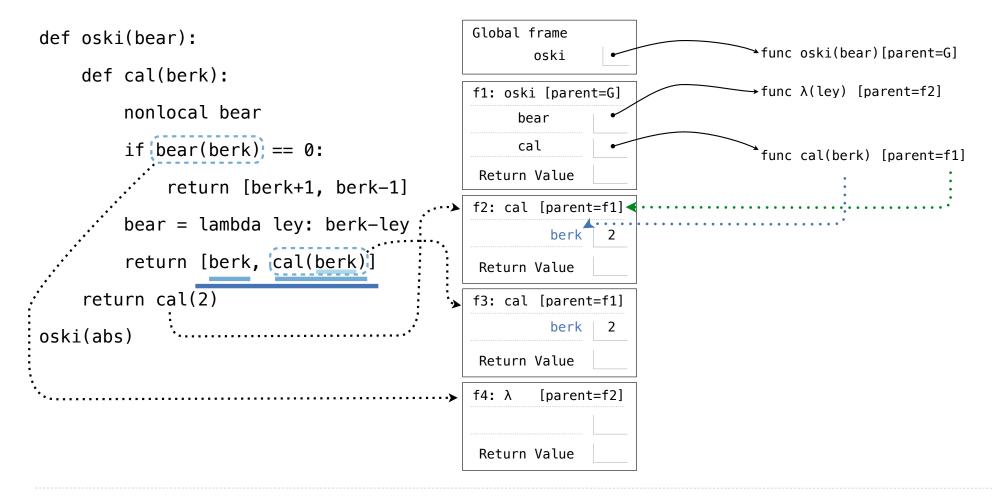
```
Global frame
def oski(bear):
                                                                                    →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                    \rightarrow func \lambda(ley) [parent=f2]
                                                   f1: oski [parent=G]
          nonlocal bear
                                                        bear
          if bear(berk) == 0:
                                                        cal
                                                                                     func cal(berk) [parent=f1]
                                                    Return Value
               return [berk+1, berk-1]
                                                   f2: cal [parent=f1]
          bear = lambda ley: berk-ley :
                                                            berk
          return [berk, cal(berk)]
                                                    Return Value
     return cal(2)
                                                   f3: cal [parent=f1]
oski(abs)
                                                    Return Value
                                                    Return Value
```

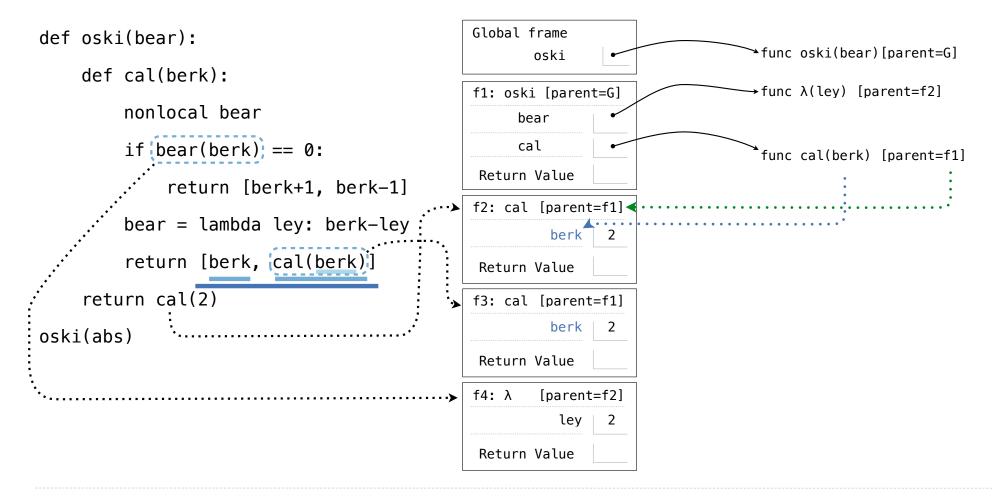
```
Global frame
def oski(bear):
                                                                                    →func oski(bear)[parent=G]
                                                          oski
     def cal(berk):
                                                                                    \rightarrow func \lambda(ley) [parent=f2]
                                                   f1: oski [parent=G]
          nonlocal bear
                                                        bear
          if bear(berk) == 0:
                                                        cal
                                                                                     func cal(berk) [parent=f1]
                                                    Return Value
               return [berk+1, berk-1]
                                                   f2: cal [parent=f1]
          bear = lambda ley: berk-ley :
                                                            berk
          return [berk, cal(berk)]
                                                    Return Value
     return cal(2)
                                                   f3: cal [parent=f1]
                                                            berk
oski(abs)
                                                    Return Value
                                                    Return Value
```

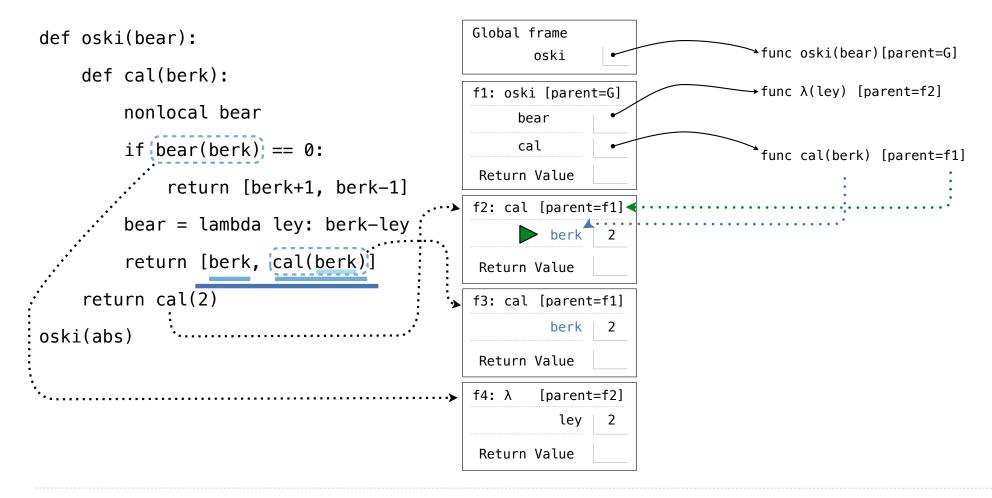
```
Global frame
def oski(bear):
                                                                                    →func oski(bear)[parent=G]
                                                           oski
     def cal(berk):
                                                                                    \rightarrow func \lambda(ley) [parent=f2]
                                                   f1: oski [parent=G]
          nonlocal bear
                                                        bear
          if bear(berk) == 0:
                                                        cal
                                                                                     func cal(berk) [parent=f1]
                                                    Return Value
               return [berk+1, berk-1]
                                                   f2: cal [parent=f1]
          bear = lambda ley: berk-ley :
                                                            berk
          return [berk, cal(berk)]
                                                    Return Value
     return cal(2)
                                                   f3: cal [parent=f1]
                                                            berk
oski(abs)
                                                    Return Value
                                                    Return Value
```

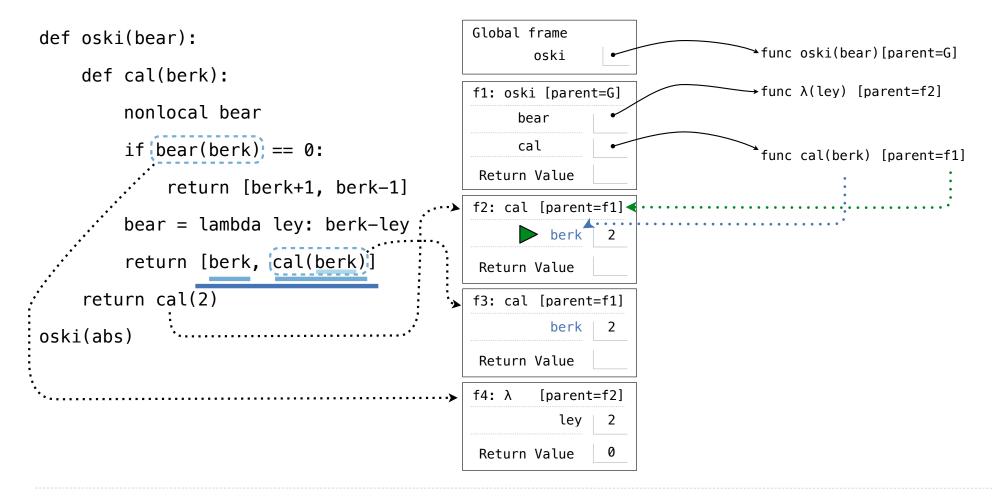


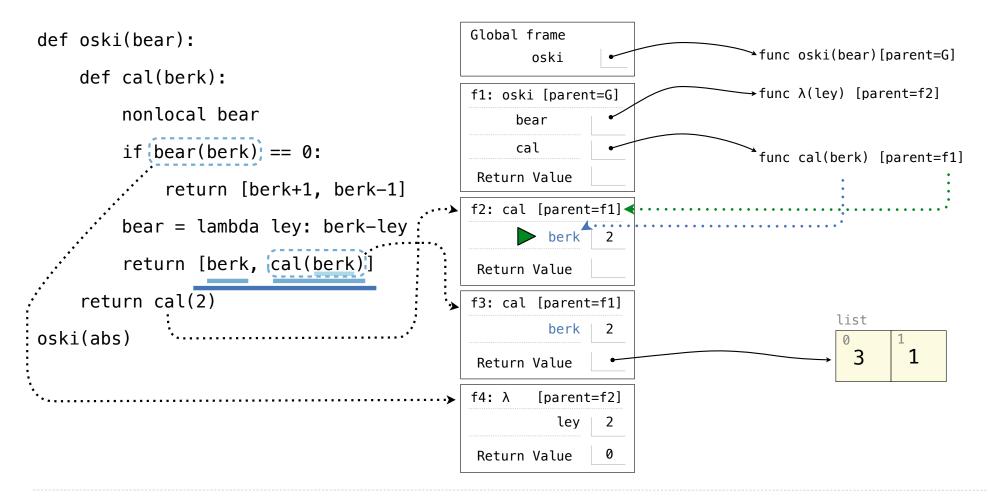


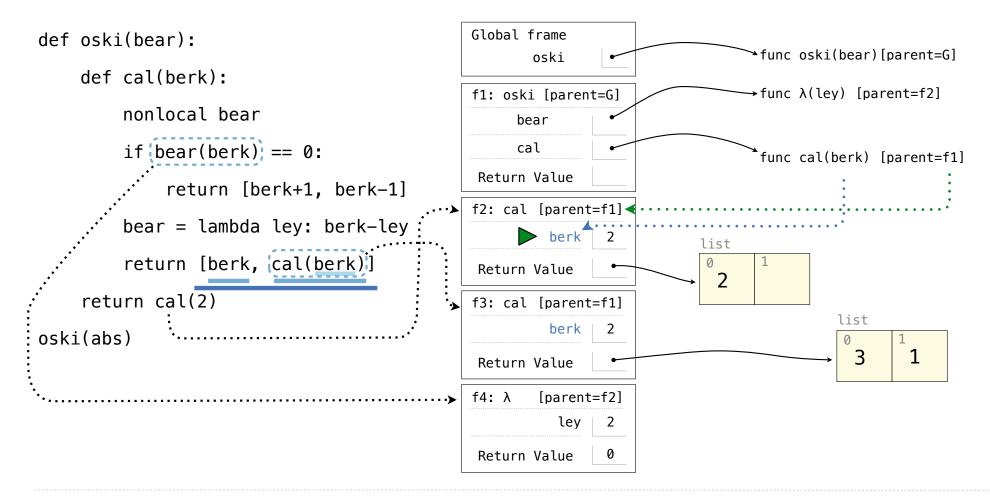


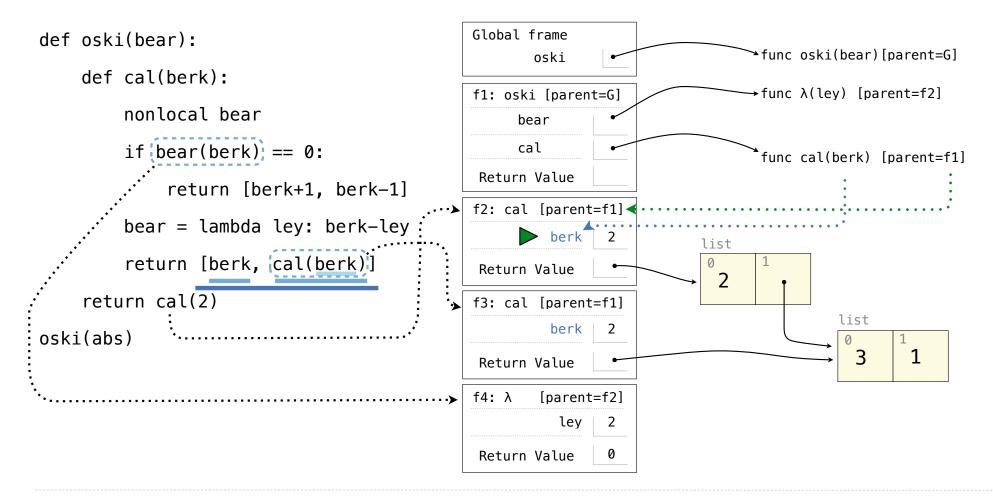


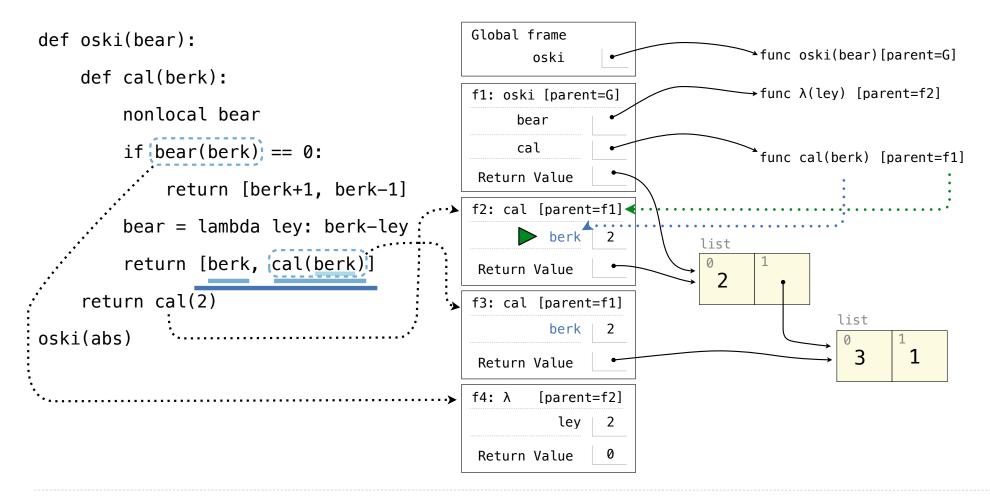


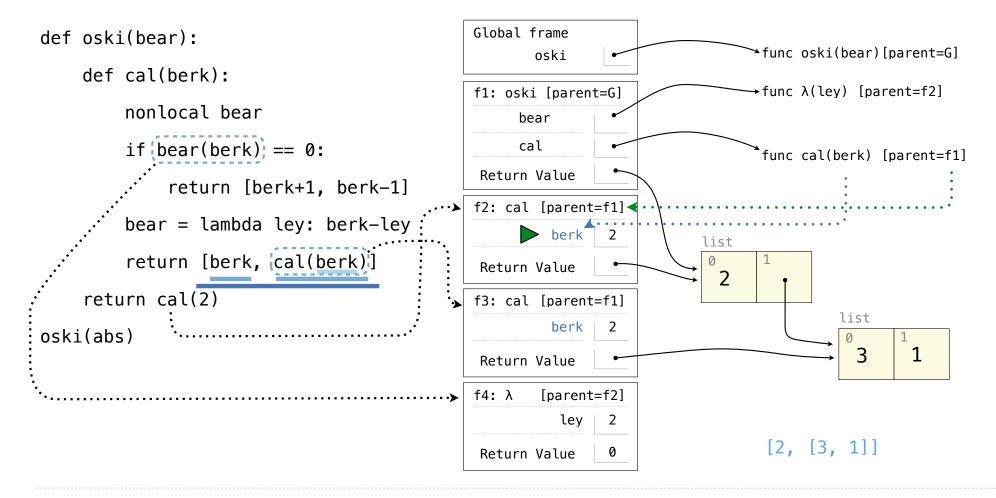


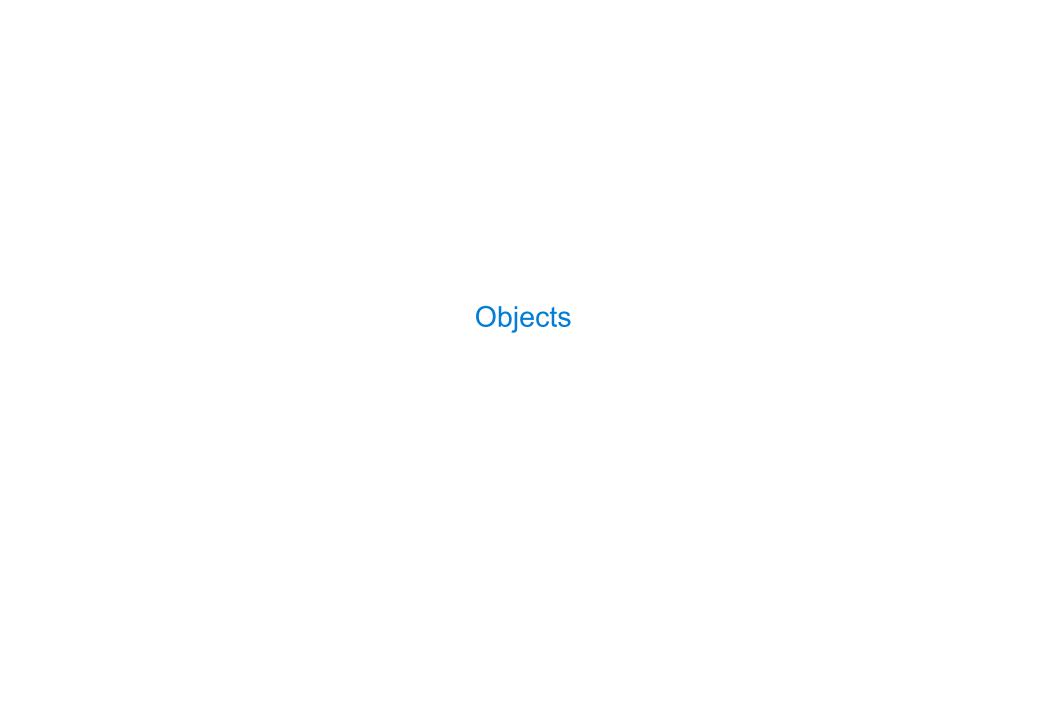












	<b>~</b> ~	0			-
La	nd	$\cup$	וועי	е	S

Lai	$\sim$	- / 1	1 A /	$\sim$	ro
			/ <b>//</b> //		
		$\sim$	V V	-	

Instance attributes are found before class attributes; class attributes are inherited class Worker:

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
```

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
class Worker:
   greeting = 'Sir'
   def init (self):
        self_elf = Worker
   def work(self):
        return self.greeting + ', I work'
   def ___repr__(self):
        return Bourgeoisie greeting
class Bourgeoisie(Worker):
    greeting = 'Peon'
   def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
class Worker:
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
   def work(self):
        return self.greeting + ', I work'
   def __repr__(self):
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john_elf_work(john)
jack = Worker()
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                         <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                          greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
   def work(self):
        return self greeting + ', I work'
   def repr (self):
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john_elf_work(john)
jack = Worker()
john = Bourgeoisie()
jack greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                          greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                          greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john_elf_work(john)
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                          greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                          greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                          elf: -
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john_elf_work(john)
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
                                                                           greeting: 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                              'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
jack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john.work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                              'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker_work(self))
                                              'I gather wealth'
        return 'I gather wealth'
                                                                           john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                              'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker_work(self))
                                              'I gather wealth'
        return 'I gather wealth'
                                                                           john <Bourgeoisie>
                                             >>> john.elf.work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self.greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                              'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker_work(self))
                                              'I gather wealth'
        return 'I gather wealth'
                                                                           john <Bourgeoisie>
                                             >>> john.elf.work(john)
iack = Worker()
                                              'Peon, I work'
                                                                           elf: -
john = Bourgeoisie()
jack.greeting = 'Maam'
```



N/	orse		0	C		
IV	w	130		しノ	u	

Morse code is a signaling protocol that transmits messages by sequences of signals

A: • **•** 

B: • • •

C: \_\_\_\_

D: • •

**E**: •

. . .

Morse code is a signaling protocol that transmits messages by sequences of signals

Problem: Implement morse so that decode works correctly

A: •

B: •••

E. •

. . .

Morse code is a signaling protocol that transmits messages by sequences of signals

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '..'}

B:
C:
D:
E:
O
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)

>>> [decode(s, t) for s in ['-..', '.', '-...', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)

>>> [decode(s, t) for s in ['-..', '.', '-...', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)

>>> [decode(s, t) for s in ['-..', '.', '-..', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)

>>> [decode(s, t) for s in ['-..', '.', '-..', '-..', '.']]
['d', 'e', 'c', 'a', 'd', 'e']
    """

for signal in signals:
        tree = [b for b in tree.branches if [b.root == signal]][0]

leaves = [b for b in tree.branches if b.is_leaf()]

assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
>>> [decode(s, t) for s in ['-..', '..', '-...', '.-.', '..']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """
for signal in signals:
        tree = [b for b in tree.branches if |b.root == signal]][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
    >>> [decode(s, t) for s in ['-..', '..', '-...', '..', '-...', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """
for signal in signals:
        tree = [b for b in tree.branches if [b.root == signal]][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
>>> [decode(s, t) for s in ['-..', '..', '-...', '...', '...', '...']]
['d', 'e', 'c', 'a', 'd', 'e']
    """
for signal in signals:
        tree = [b for b in tree.branches if b.root == signal][0]
leaves = [b for b in tree.branches if b.is_leaf()]
assert len(leaves) == 1
return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-...', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
    >>> [decode(s, t) for s in ['-..', '..', '-...', '.-.', '..']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    """

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)
>>> [decode(s, t) for s in ['-..', '.', '-...', '.-.', '...']]
['d', 'e', 'c', 'a', 'd', 'e']
    """

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
leaves = [b for b in tree.branches if b.is_leaf()]
assert len(leaves) == 1
return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly

abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-...', 'e': '.'}

def decode(signals, tree):
    """Decode signals into a letter.

>>> t = morse(abcde)

>>> [decode(s, t) for s in ['-...', '...', '-...', '...']]
    ['d', 'e', 'c', 'a', 'd', 'e']

"""

for signal in signals:
    tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is_leaf()]
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}
def decode(signals, tree):
    """Decode signals into a letter.
                                      def morse(code):
                                                                                     ?
    >>> t = morse(abcde) +----
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', ['.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    for signal in signals:
        tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is leaf()]
                                                                          'e'
                                                                                 I \subseteq I
    assert len(leaves) == 1
    return leaves[0].root
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}
def decode(signals, tree):
    """Decode signals into a letter.
                                    def morse(code):
                                                                                ?
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', ['.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
   for signal in signals:
       tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is leaf()]
                                                                      ا ۾ ا
                                                                             I \perp I
    assert len(leaves) == 1
    return leaves[0].root
                                                                             'a'
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}
def decode(signals, tree):
    """Decode signals into a letter.
                                    def morse(code):
                                                                                ?
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', ['.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
   for signal in signals:
       tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is leaf()]
                                                                     ا ۾ ا
                                                                             I = I
    assert len(leaves) == 1
    return leaves[0].root
                                                                             'a'
```

```
Problem: Implement morse so that decode works correctly
abcde = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.'}
def decode(signals, tree):
    """Decode signals into a letter.
                                      def morse(code):
                                                                                    ?
    >>> t = morse(abcde) +----
    >>> [decode(s, t) for s in ['-..', '.', '-.-.', ['.-', '-..', '.']]
    ['d', 'e', 'c', 'a', 'd', 'e']
    for signal in signals:
        tree = [b for b in tree.branches if b.root == signal][0]
    leaves = [b for b in tree.branches if b.is leaf()]
                                                                         ا ۾ ا
                                                                                 I = I
    assert len(leaves) == 1
    return leaves[0].root
                                                                                 'a'
                                           (Demo)
```