# Nested function parenting

What happens when this script is executed?

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhowmuch
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

### Nested fun (#0 <global>

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhowmuch
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder #0
```

### Nested fun (#0 <global>

```
def make_adder(byhowmuch)
    def this_add(x):
        return x + byh
    return this_add
add1 = make adder(1)
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
fin make_adder #0

#1 make_adder #0

byhowmuch 1
```

## Nested fun (#0 <global>

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
fn make_adder #0

#1 make_adder #0

byhowmuch 1

this_add  
fn this_add #1
```

What happens when this so

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
fn make_adder #0

#1 make_adder #0

byhowmuch 1

this_add  
fn this_add #1
```

→ A new this\_add function object is constructed during this call.

```
def make_adder(byhowmuc
    def this_add(x):
        return x + byho
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
fin make_adder #0

byhowmuch 1

this_add  
fin this_add #1
```

- → A new this\_add function object is constructed during this call.
- → The *def* is executed within the context of frame #1.

```
def make_adder(byhowmuch)
    def this_add(x):
        return x + byhow
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
#1 make_adder #0
byhowmuch 1
this_add  
fn this_add #1
```

- → A new this\_add function object is constructed during this call.
- → The *def* is executed within the context of frame #1.
- → That new function's parent frame is set to #1.

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
#1 make_adder #0

byhowmuch 1

this_add  
fn this_add #1
```

- → A new this\_add function object is constructed during this call.
- → The *def* is executed within the context of frame #1.
- → That new function's parent frame is set to #1.

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

```
#0 <global>
make_adder  
fn make_adder #0

add1

#1 make_adder #0

byhowmuch 1

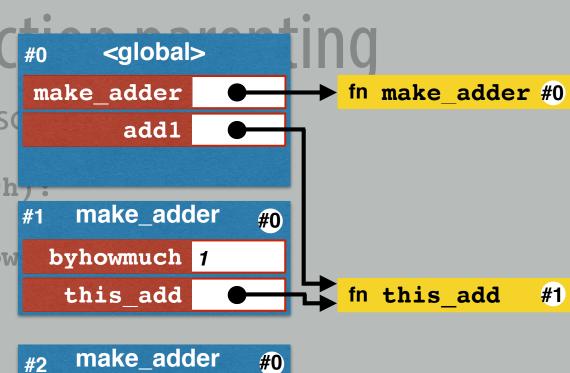
this_add  
fn this_add #1
```

What happens when this so

```
dof make_adder(byhowmuch, def this_add(x):

return x + byhow
return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



byhowmuch 5

```
def make_adder(byhowmuch)
    def this_add(x):
        return x + byhow
    return this add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

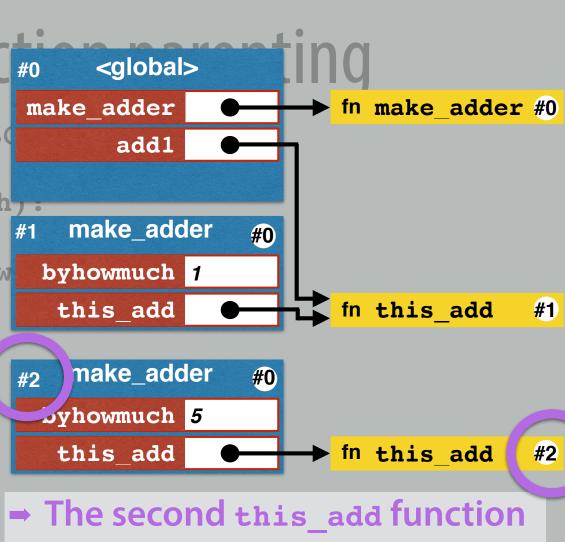
```
<global>
make adder
                      fn make adder #0
       add1
   make_adder
#1
 byhowmuch 1
                        fn this_add
   this add
                                      #1
    make_adder
                 #0
#2
  syhowmuch 5
   this_add
                      fn this add
                                      #2
```

- → A second this\_add function object is constructed during this call with parameter 5.
- → The *def* is executed within the context of frame #2.

What happens when this so

```
def make adder(byhowmuch)
    def this_add(x):
        return x + byhow
    return this add
```

```
add1 = make adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



object's parent frame is #2.

What happens when this so

```
def make_adder(byhowmuch
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
    add5 = make_adder(5)
```

print(add1(100))

print(add5(200))

```
<global>
make adder
                       fn make adder #0
       add1
       add5
   make_adder
#1
 byhowmuch 1
                        fn this_add
   this_add
                                      #1
   make_adder
                 #0
#2
 byhowmuch 5
   this_add
                      fn this add
                                      #2
```

What happens when this so

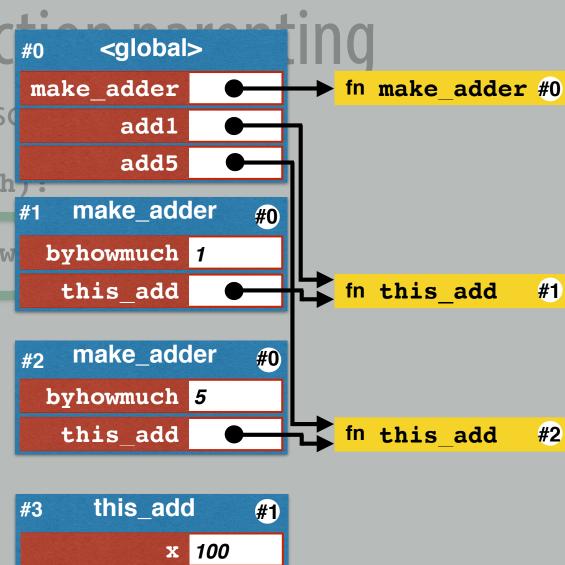
print(add5(200))

```
<global>
make adder
                      fn make adder #0
       add1
       add5
   make_adder
#1
 byhowmuch 1
                       fn this_add
   this add
                                     #1
   make_adder
                 #0
 byhowmuch 5
   this add
                     fn this add
                                     #2
```

- Now the top-level code has access to two different adding functions add1 and add5.
- → Each is a different closure.

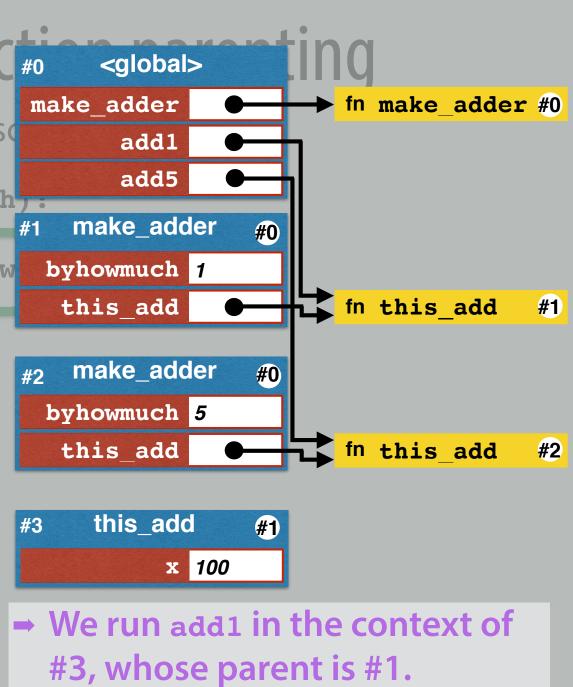
```
def make_adder(byhowmuch)
    def this_add(x):
        return x + byhow
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



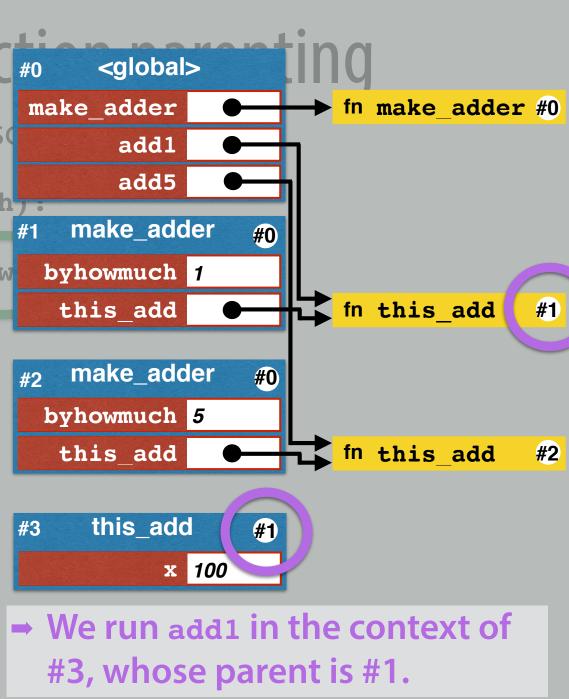
```
def make_adder(byhowmuch)
    def this_add(x);
    return x + byhow
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



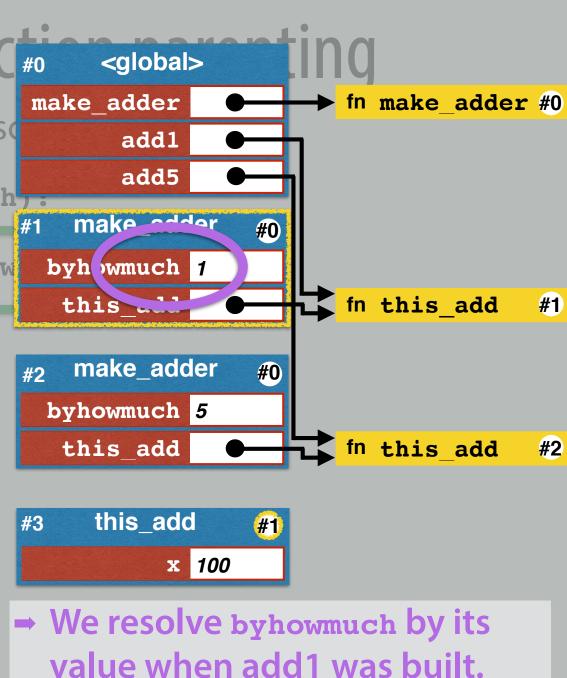
```
def make_adder(byhowmuch)
    def this_add(x);
    return x + byhow
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



```
def make_adder(byhowmuch_
    def this_add(x);
    return x + byhow
    return this_add
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



What happens when this so

```
def make_adder(byhowmuch
    def this_add(x):
    return x + byhow
    return this_add

add1 = make_adder(1)
    add5 = make_adder(5)
    print(add1(100))
#2
```

print(add5(200))

```
<global>
make adder
                        fn make adder #0
       add1
       add5
    make_adder
#1
 byhowmuch 1
                        fn this_add
   this_add
                                       #1
    make_adder
                  #0
#2
 byhowmuch 5
   this_add
                        fn this add
                                       #2
     this_add
#3
                  #1
           X 100
```

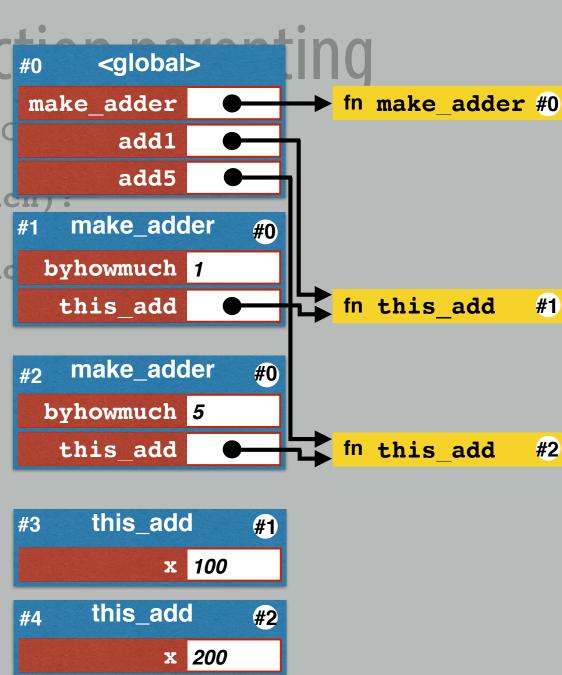
What happens when this so

```
def make_adder(byhowmuc)
    def this_add(x):
        return x + byho by
    return this_add
    t
```

```
add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

Output to the console:

```
101205
```



# Following the parent chain

When a name is accessed in a statement:

- → Python checks the active local frame.
- → If not there, Python checks its parent frame.
- → If not there, Python checks its parent's parent frame.
- → Etc.
- ➡ Eventually this could hit the global frame, raise an error.

This is the *execution environment* of that statement.

# Frames can get complicated.

What happens when this script is executed?

```
apply2 = (lambda f: (lambda x: f(f(x))))
def make adder(byhowmuch):
    return (lambda x: x + byhowmuch)
add1 = make adder(1)
add5 = make adder(5)
this = apply2(add1)
that = apply2(add5)
this(1000)
that(1000)
```

# **Environment Example**

Show this code's execution with an environment diagram:

```
def thing_maker(a,b,c):
    x = 35+b
    c = c+1
    def thing(v,w):
        print(x)
        print(a)
        u = v*b + w
        return u
    x = x - 30
    return thing
thingA = thing maker(3,4,5)
r = thingA(10,11)
print(r)
```