

```
>>> # What if we define the same variable twice?
>>> x = 10
>>> x
10
>>> x = 11
>>> x
11
>>> x = x + 1
>>> x
12
>>> y = x + 100
>>> y
112
>>> x = x + 1
>>> x
13
>>> y
112
```

New: updating variables

within each **function call** and at the **top level**, Python keeps track of a mapping between variable names and their values that can change with a variable update or variable assignment. This mapping is called an **environment** or a **scope**.

x = <expression>

If x is **not** defined already in the current **scope**, add it with the value of <expression>.

If x **is** defined already in the current scope, evaluate <expression>, then **change** the value of x to that value in the current scope.

```
== RESTART
>>> abs(-10)
10
>>> n
NameError: name 'n' is not defined
>>> abs2(-100)
100
>>> n
>>> abs3(-10)
10
>>> m
100
```

```
def abs(n):
    if n < 0: return n * -1
    else: return n

def abs2(n):
    if n < 0:
        n = n * -1
    return n

m = 100
def abs3(m):
    if m < 0:
        m = m * -1
    return m
```

```
abs3(-10)
print(m)
```

m: 100

```
if m < 0:
    m = m * -1
return m
```

m: -10

m: 100

```
print(m)
```

```
if True:
    m = 10
return m
```

m: -10

m: 100

```
print(m)
```

```
return m
```

m: 10

m: 100

```
print(m)
```

```
10
print(m)
```

m: 100

```
10
print(100)
```

m: 100

Create the inner scope for evaluating the function call. Track a separate m for the duration of the function call evaluating.

The condition is True, so evaluate the then branch. This is a variable update to m.

It updates the value of the m variable **for this scope, and only this scope**.

When we return m, we use the value for this scope (10).

Once the function is finished, we continue evaluating in the top-level scope.

```

== RESTART: ...
22
37
3
4
>>> total
66

```

```

lst = [22, 37, 3, 4]
for n in lst:
    print(n)

```

```

sum = 0
for m in lst:
    sum = sum + m

```

write a function product that takes a list of numbers
and returns the product of all the numbers multiplied

```

assert product([2, 3, 4]) == 24

```

```

sum = 0
for m in lst:
    sum = sum + m

```

```

lst: [22, 37, 3, 4]
sum: 0

```

```

sum = 0
for m in lst:
    sum = sum + m

```

```

lst: [22, 37, 3, 4]
sum: 0
m: 22

```

```

sum = 0
for m in lst:
    sum = 0 + 22

```

```

lst: [22, 37, 3, 4]
sum: 22
m: 22

```

```

sum = 0
for m in lst:
    sum = sum + m

```

```

lst: [22, 37, 3, 4]
sum: 22
m: 37

```

```

sum = 0
for m in lst:
    sum = 22 + 37

```

```

lst: [22, 37, 3, 4]
sum: 59
m: 37

```

repeat until done with the list!

```

Item 0 is: 22
Item 1 is: 37
Item 2 is: 3
Item 3 is: 4
>>> list(range(0, 7))
[0, 1, 2, 3, 4, 5, 6]
>>> list(range(5, 9))
[5, 6, 7, 8]
>>> list(range(0, len(lst)))
[0, 1, 2, 3]

```

```

lst = [22, 37, 3, 4]

```

```

for index in range(0, len(lst)):
    print("Item " + str(index) + " is: " + str(lst[index]))

```

Write a function factorial that takes a number n
and returns n * (n - 1) * (n - 2) * ... * 1

Write a function numbered_list that takes a list
of strings strs and produces a string formatted
in a numbered list like "1. <string1> 2. <string2>"