Functions

Rich Enbody

From Mathematics we know that *functions* perform some operation and return one value.

Functions “encapsulate”
the performance of some operation,
so it can be used by others.

(for example, the sqrt() function)

Consider a *function*
to convert Celsius to Fahrenheit:

Formula: F = C * 1.8 + 32.0

Functional notation:

F = g(C) where g(C) = C*1.8 + 32.0

Invocation

Math: F= g(C)

Python: F = g(C)

Terminology: argument “C”

Definition

Math: g(C) = C*1.8 + 32.0

Python:

```
def g(C):
    return C*1.8 + 32.0
```

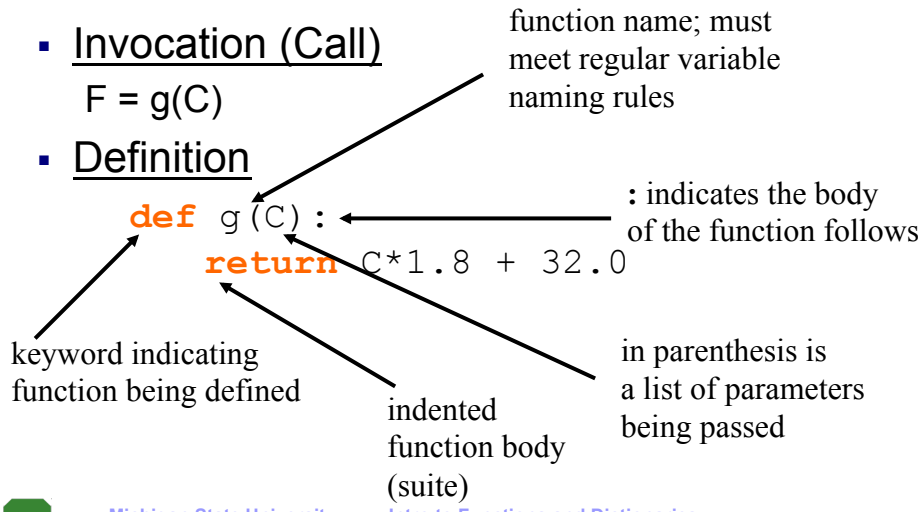
Terminology: parameter “C”

Terminology

- Invocation (Call)

  F = g(C)

- Definition



Verbose Python Invocation

Math: F= g(C)

Python:

```
cels_temp = 100
fahr_temp = celsius_to_fahrenheit(cels_temp)
print(cels_temp,'in Fahrenheit is:',fahr_temp)
```

Verbose Python Definition

Math: g(C) = C*1.8 + 32.0
Python:

```
def celsius_to_fahrenheit(cels_temp):
    return cels_temp*1.8 + 32.0
```

Program Abstraction

1. Get Temperature
2. Convert Temperature
3. Display Temperature

Needed Behavior

```
Please enter a temperature in
  degrees Celsius: 19.5
  Original:     19.5 C
  Equivalent:   67.1 F
```

Get a temperature

```
def get_temp():
    cels_temp = input(\
        "Please enter temp in Celsius:")
    return float(cels_temp)
```

Convert

```
def celsius_to_fahrenheit(cels_temp):
    result = cels_temp*1.8+32.0
    return result
```

Display results

```
def display(cels_temp,fahr_temp):
    print("Original: ",cels_temp)
    print("Equivalent:",fahr_temp)
    print()
```

Exercise

Write a function named vowel_count that takes a string as a parameter and returns a count of the number of vowels in the string.

```
def vowel_count(str1):
```

The function arguments
are passed *in order*
to the function parameters.

The names need not match.

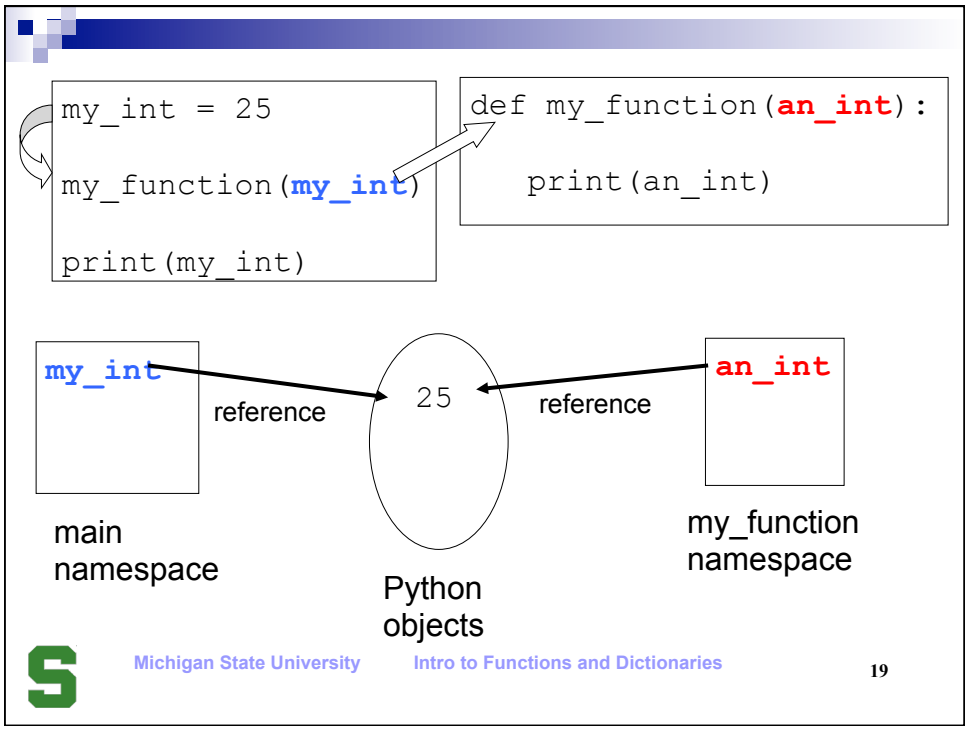
When a function runs,
it defines a new namespace.

The names in the function's namespace
are only available to the function.

Passing arguments by reference:

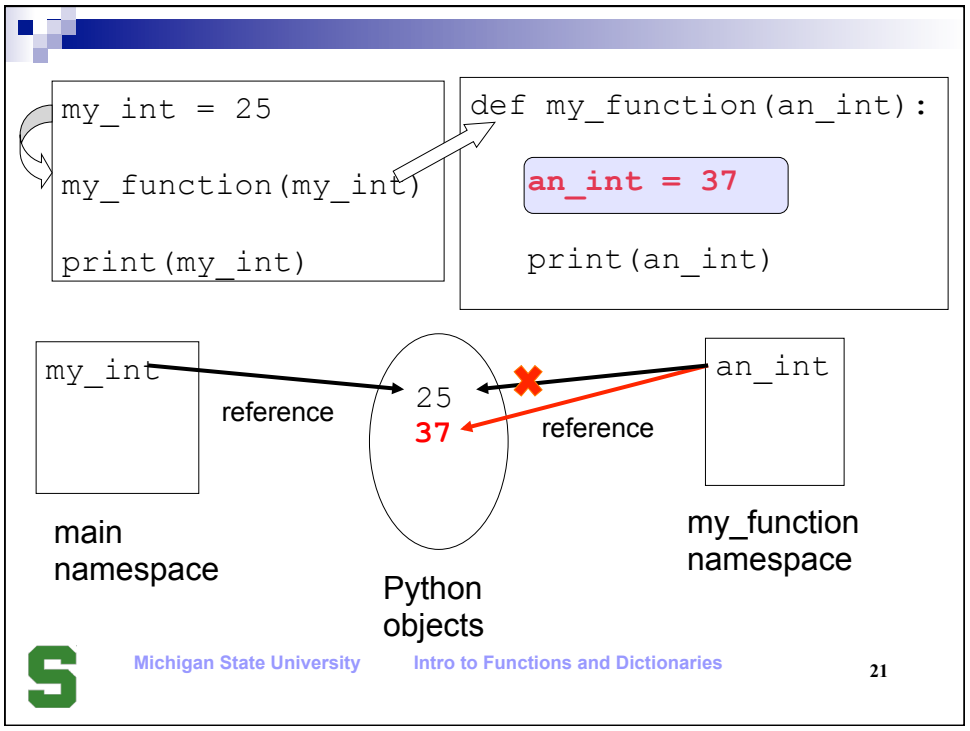
the first argument
passes its **namespace reference**
to the first parameter,
second to second, and so on.

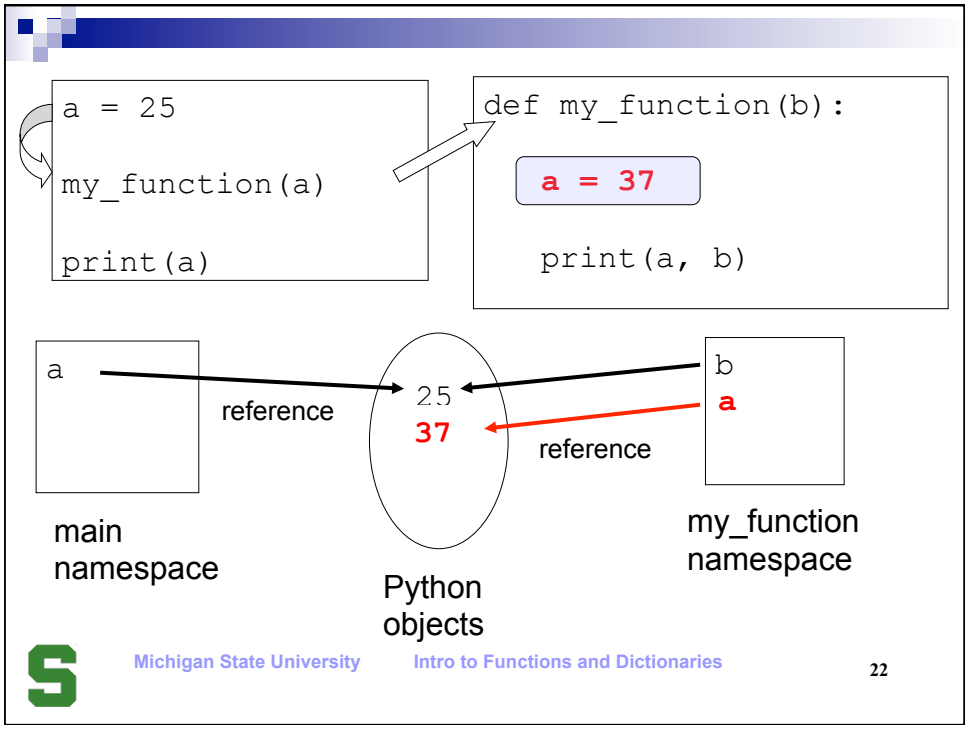
What gets passed?



Both namespaces now refer to the same object: the reference got passed.

What happens
if the function changes that variable?


my_int = 25
my_function(my_int)
print(my_int)
def my_function(an_int):
an_int = 37
print(an_int)
my_int
reference
25
37
reference
an_int
main
namespace
Python
objects
my_function
namespace

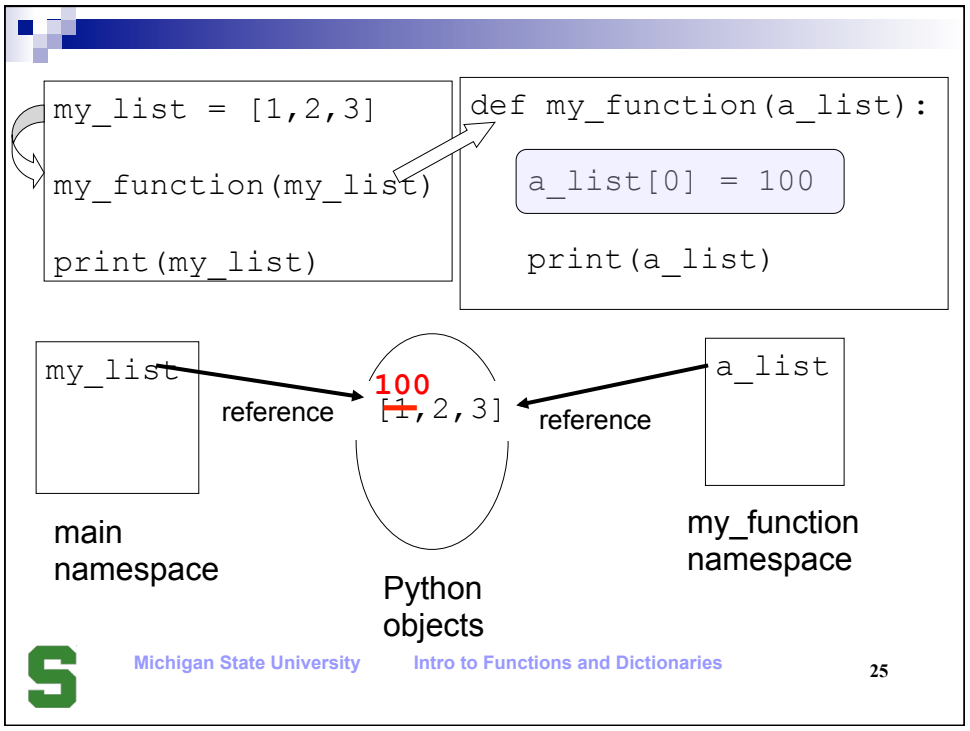

a = 25
my_function(a)
print(a)
def my_function(b):
a = 37
print(a, b)
a
reference
25
37
reference
b
a
main
namespace
Python
objects
my_function
namespace

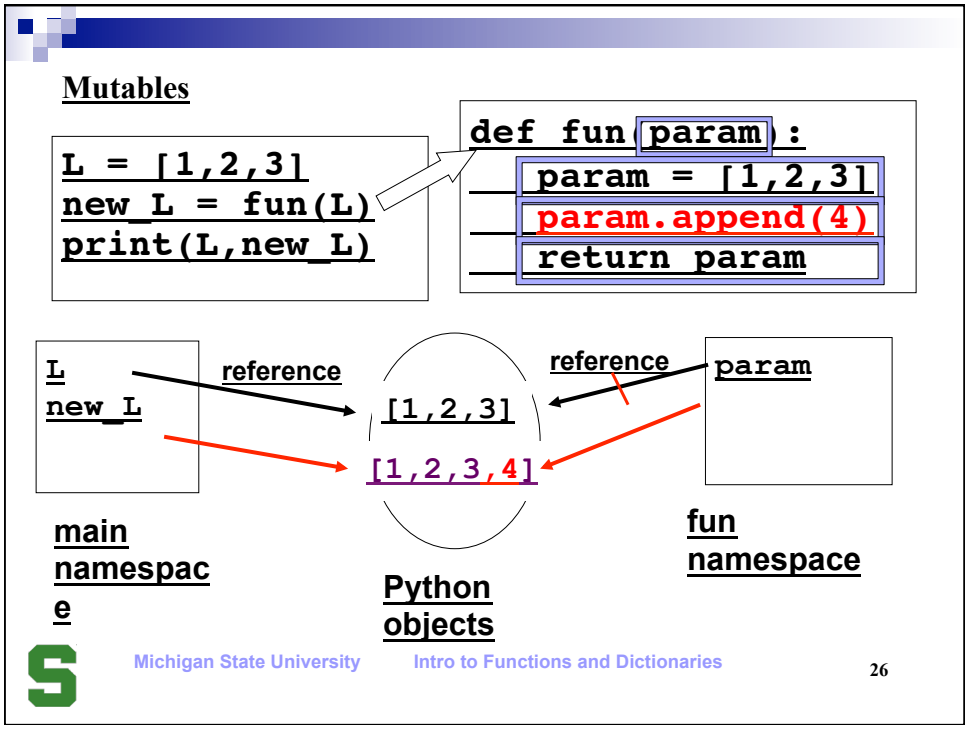
The object (int) is immutable:
it wasn't changed.

The function namespace simply updated its reference to a new object.

The reference in the calling program was unaffected.

What happens with mutables such as lists?

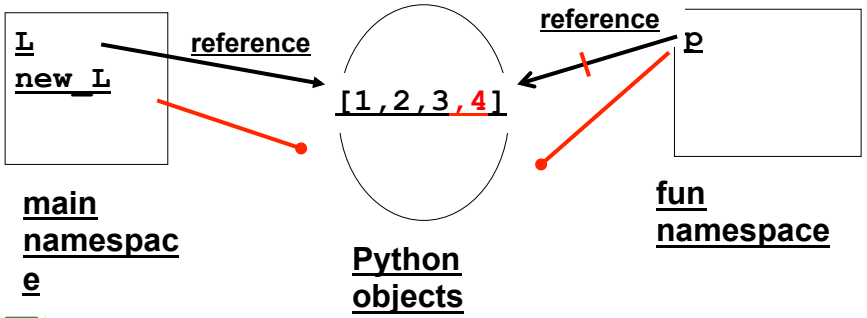

my_list = [1,2,3]
my_function(my_list)
print(my_list)
def my_function(a_list):
a_list[0] = 100
print(a_list)
my_list
reference
100
[1,2,3]
reference
a_list
main namespace
Python objects
my_function namespace


Mutables
L = [1,2,3]
new_L = fun(L)
print(L,new_L)
def fun(param):
param = [1,2,3]
param.append(4)
return param
L
new_L
reference
[1,2,3]
[1,2,3,4]
reference
param
main namespace
Python objects
fun namespace

Mutables

```
L = [1,2,3]
new_L = fun(L)
print(L,new_L)
```

```
def fun(p):
    p = p.append(4)
    return p
```



Exercise

Replace item at index “i” with “a_value” if “i” is a valid index.

If “i” is one greater than the maximum valid index, append “a_value” to “a_list”

If successful, return True, else return False.

```
def update( a_list, i, a_value ):


A = [ 1.2, 3.4, 5.6 ]
flag = update( A, 1, 9999 )
print( flag, A )     # Displays  True [1.2, 9999, 5.6]
flag = update( A, 3, 8888 )
print( flag, A )     # Displays  True [1.2, 9999, 5.6, 8888]
flag = update( A, 20, 7777 )
print( flag, A )     # Displays  False [1.2, 9999, 5.6, 8888]
```

Function “rules of thumb”

- Should do one thing.
  A function *abstracts* one idea.
- Should not be overly long
  (~one page of code).
- Best if generic
  so it could be reused elsewhere.

Default and Named parameters

```
def box(height=10,width=10,depth=10,
        color= "blue“ ):
          ... do something ...
```

The parameter assignment means two things:

- If the caller does not provide a value,
  the default is the parameter’s assigned value.
- You can get around the order of parameters
  by using the name.

Defaults

```
def box (height=10,width=10,length=10):
     print(height,width,length)

box()    # prints 10 10 10
box(20)  # prints 20 10 10
box(20,30)
```

Named parameter

```
def box (height=10,width=10,length=10):
     print(height,width,length)

box(length=25,height=25)
  # prints 25 10 25

box(15,15,15)     # prints 15 15 15
```