References

CMPT 145

Learning Objectives

After studying this chapter, a student should be able to:

- Explain the difference between a value and an object.
- Explain what a reference is.
- Explain how Python uses frames and references to associate variables and values.
- Explain how Python evaluates expressions involving mutable and immutable values.
- Draw diagrams showing the frame(s), values, and objects, given a sequence of Python expressions or statements.
- Explain how frames are used for a function's local variables.
- Explain what happens when a local variable shadows a global variable.
- Explain the difference between == and `is`.
- Explain the difference between copying references and values.

Data values and objects

A data value represents information in a script.

1. Numbers, strings, `True`, etc
2. Data values appear in Python scripts.
3. Usually displayed or written for humans to read easily.

Python represents all data values as objects.

- Objects are stored inside the computer, in a region of memory called the heap.
- Python stores the data using its rules to create objects.
- You'll learn more about these ideas in 200-level classes.

Addresses and References

Every object in the heap has an address.

- The address is used by Python to locate the object.
- The address is a also used as the object's identity.
- The address is a reference to the object.

Expressions and commands

In Python:

- Evaluating an expression creates an object.
  - E.g, `3 + 4` creates a new number
  - E.g., `[3] + [4]` creates a new list
- Executing a command has an effect on something that already exists.
  - E.g., `print(7)` sends data to the console.
  - E.g., `alist.append(3)` adds a new value to the list.

Variables

A variable has 3 aspects:

1. Its name
2. Its value
3. Its address

Variables and Frames

In Python:

- Variables are kept in a table called a frame
- A frame associates a variable name with its value using an address
- Frames are managed by Python runtime system.

References

In Python:

- A reference is an address, which gives the location of an object in memory.
- We can only manipulate addresses by assignment statements.
- It is more helpful to think of a reference as an arrow to an object.

Assignment statements

```
avar = expr
```

- An assignment statement has the following effect
  1. The expression `expr` is evaluated, creating an object.
  2. The reference to the object is stored in the frame beside `avar`
- When a variable is assigned for the first time:
  1. Its name is added to the frame
  2. The reference is added to the frame

Python treats all data the same

Assignment statements never make copies of any objects.

Assignment statements only copy references.

Some languages treat data differently

- In second year, you'll learn C/C++ and Java.
- When you get there, you'll need to distinguish between simple data and compound data.
- In C/C++, Java (C-like languages):
    - Assignment statements using simple data copies the simple data.
    - Assignment statements using compound data copies the references.
    - Why? Speed.

Frame diagrams

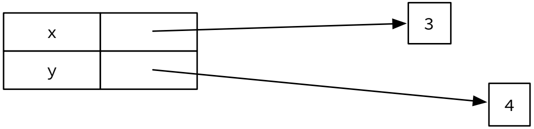
- Python code:

```python
x = 3
y = 4
```

- Simplified frame representation by Python interpreter:

x	0x10397e7b8
y	0x10397e840

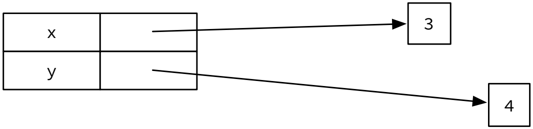
- Frame Diagram



Frame Diagram conventions

Numbers, strings, Booleans, `None` are drawn with a box.

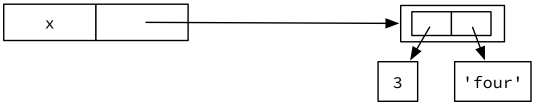
```
x = 3
y = 4
```



Frame Diagram conventions

Lists are drawn as a stretched box with segments.

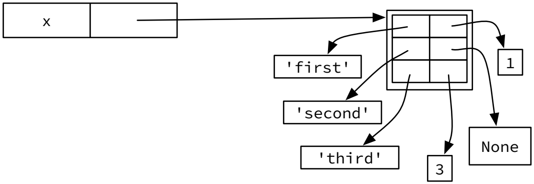
```
x = [3, 'four']
```



Frame Diagram conventions

Dictionaries are drawn as a rectangle with rows and columns.

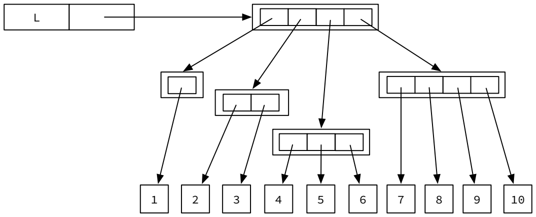
```
x = {'first':1, 'second':None, 'third':3}
```



Frame Diagram conventions

Nested structures have references to other structures.

```
L = [[1], [2,3], [4,5,6], [7,8,9,10]]
```



Python equality

In Python:

- `x == y` is `True` if the values are equal.
- `x is y` is `True` if the references are equal.

Functions and Frames

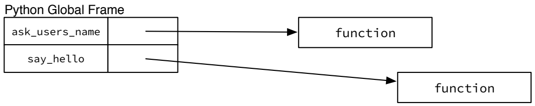
In Python:

- Calling a function creates a new frame.
- The function's parameters are variables in the frame.
- The parameter's values are copies of addresses of the arguments in the function call.
- New variables in the body of the function are added to the (new) frame.

Frame diagrams with functions

Function definitions create named function objects.

```
def ask_users_name(greeting):
    name = input(greeting)
    return name

def say_hello(user):
    print('Hello', user)
    return
```

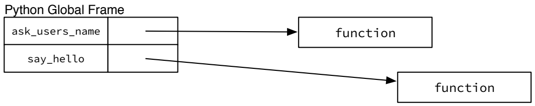


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```
aname = ask_users_name("What's your moniker? ")
say_hello(aname)
```

Before Line 1, we have:

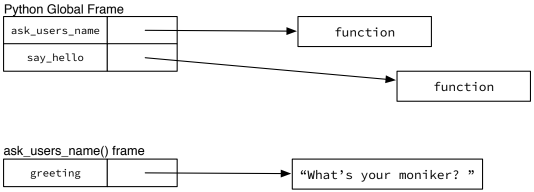


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```
aname = ask_users_name("What's your moniker? ")
say_hello(aname)
```

Calling `ask_users_name()` on Line 1 creates a new frame:

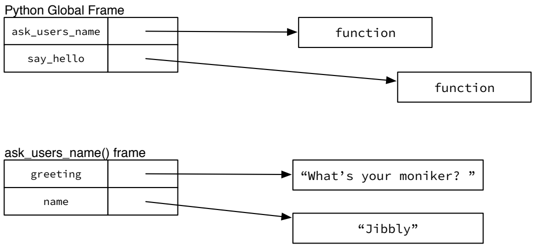


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```
aname = ask_users_name("What's your moniker? ")
say_hello(aname)
```

Just before `ask_users_name()` returns:

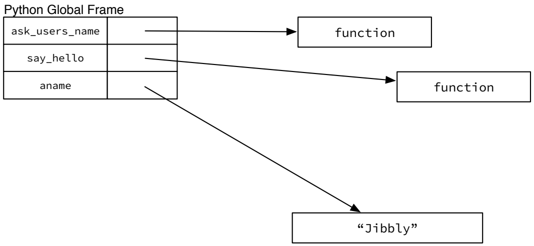


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```
aname = ask_users_name("What's your moniker? ")
say_hello(aname)
```

Finished the assignment statement on line 1, just after `ask_users_name()` returned:

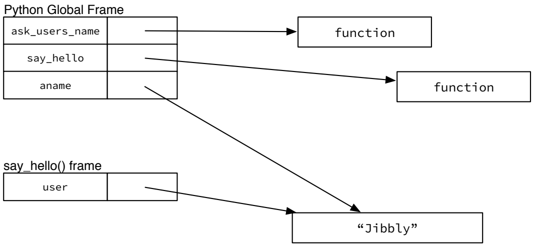


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```python
aname = ask_users_name("What’s your moniker? ")
say_hello(aname)
```

Calling `say_hello()` on Line 2 creates a new frame:

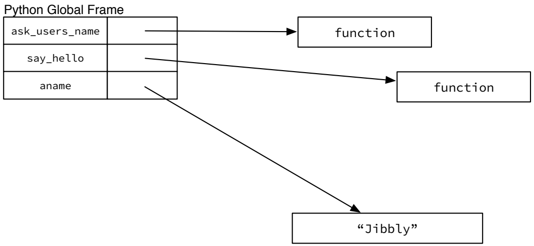


Frame diagrams with functions

Function calls dynamically create frames. A sequence of diagrams is needed.

```
aname = ask_users_name("What's your moniker? ")
say_hello(aname)
```

After `say_hello()` returned:



Caching common values

In Python:

- Very common values are created when Python starts.
- E.g., `None`, `True`, small integers.
- Expressions do not create new objects with these values.
- Caching helps increase Python efficiency.

Figuring this all out

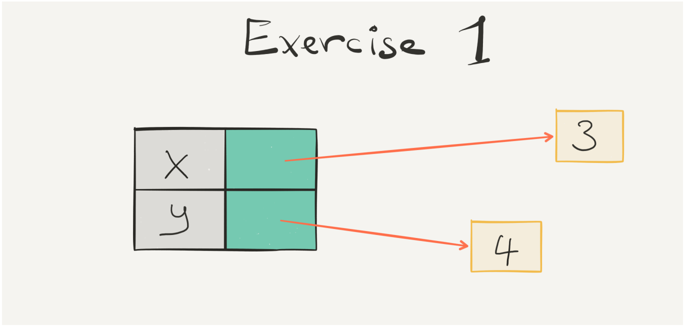
None of this is actually difficult. There are good reasons why it's confusing:

- You don't know why this is important. Yet.
  - We need it for Chapters 13, 16, 17, 20, 21.
  - You will need it for all CMPT courses from now on.
  - Every programming language uses the concept of reference; there are some differences in detail.
- None of this is visible; it's all part of the Python interpreter's work inside the computer.
  - Misusing references is probably the #2 source of software bugs.

Example 1

Draw a diagram that shows the frames, variables, values and references for the following Python code:

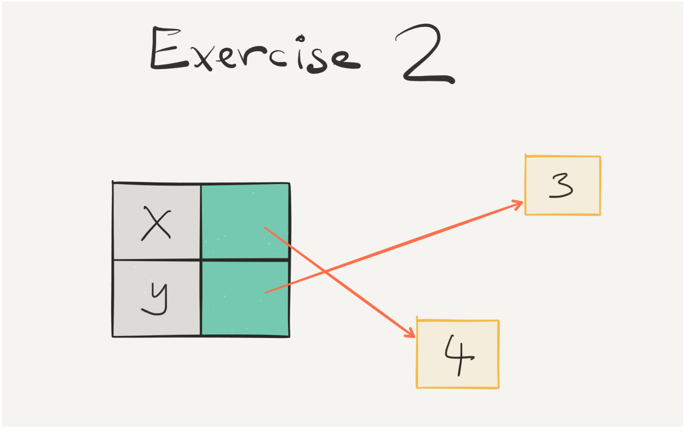
```
x = 3
y = x + 1
```


Exercise 1
x
y
3
4

Example 2

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```
x = 3
y = x
x = 4
```

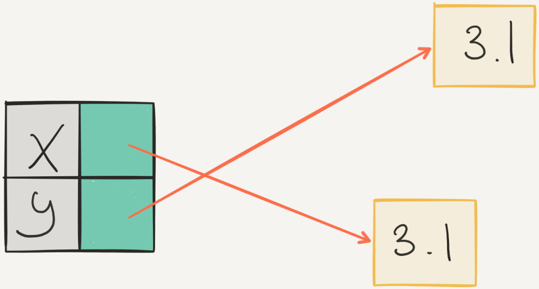

Exercise 2
x
y
3
4

Example 3

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
x = 3.1
y = x  + 0.0
print(x == y)
print(x is y)
```

Exercise 3



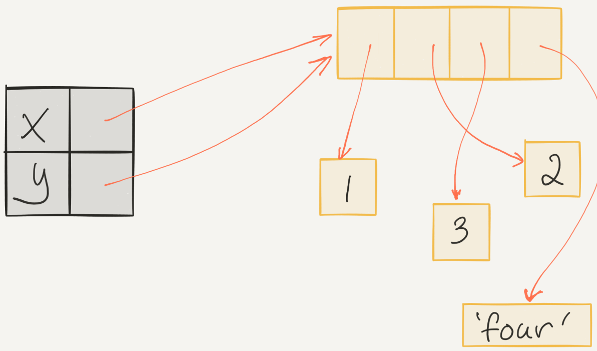
```
>>> x = 3.1
>>> y = x + 0.0
>>> print(x == y)
True
>>> print(x is y)
False
```

Example 4

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
x = [1, 2, 3]
y = x
y.append('four')
```

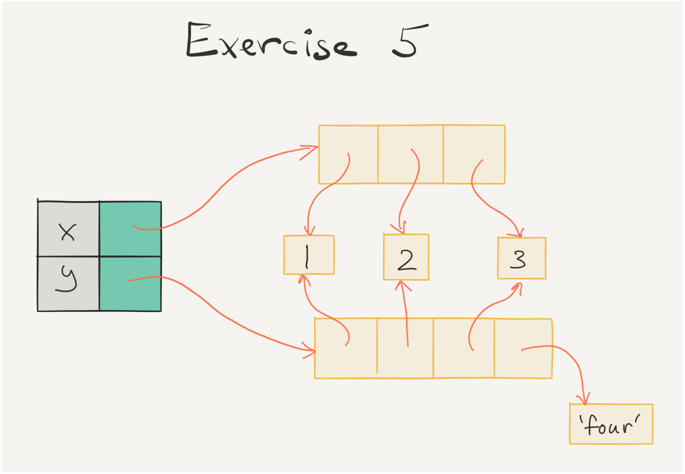
Exercise 4



Example 5

Draw a diagram that shows the frames, variables, values and references for the following Python code:

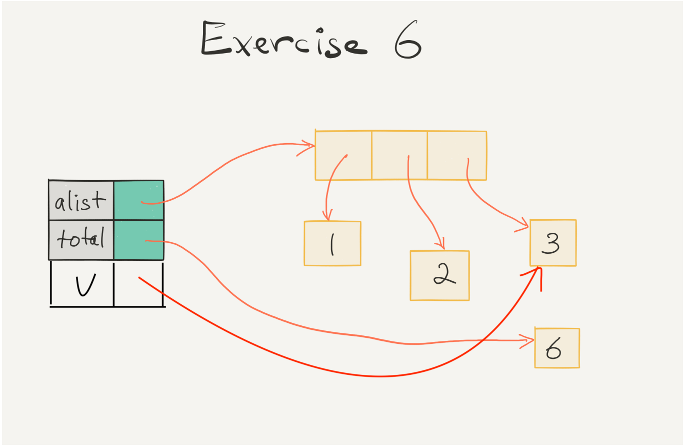
```python
x = [1, 2, 3]
y = x + ['four']
```


Exercise 5
x
y
1
2
3
'four'

Example 6

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
alist = [1,2,3]
total = 0
for v in alist:
    total = total + v
```

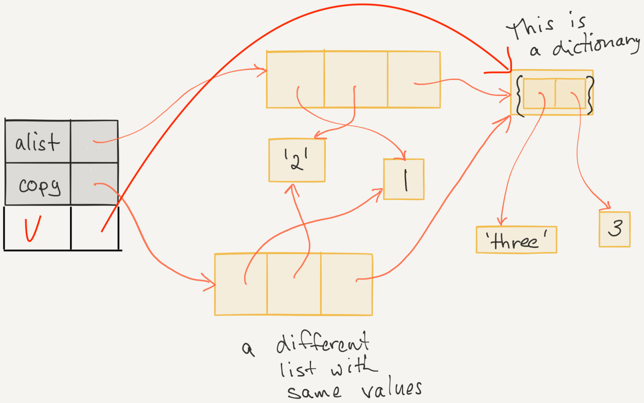

Exercise 6
alist
total
V
1
2
3
6

Example 7

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
alist = [1, '2', {'three':3}]
copy = []
for v in alist:
    copy.append(v)
```

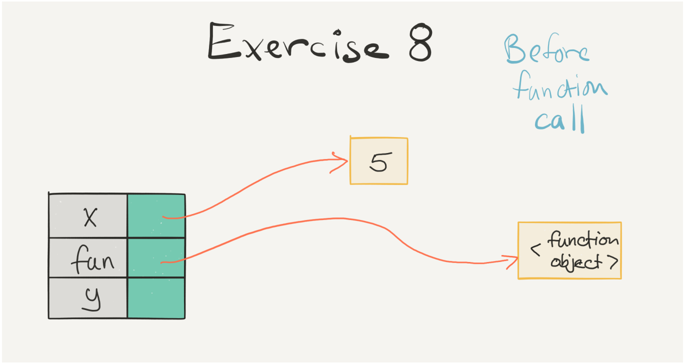
Exercise 7

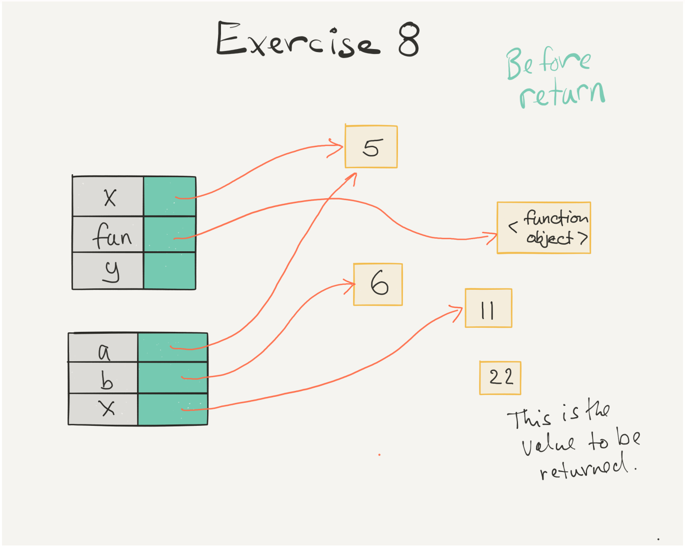

this is a dictionary
alist
copy
V
'2'
1
'three'
3
a different list with same values

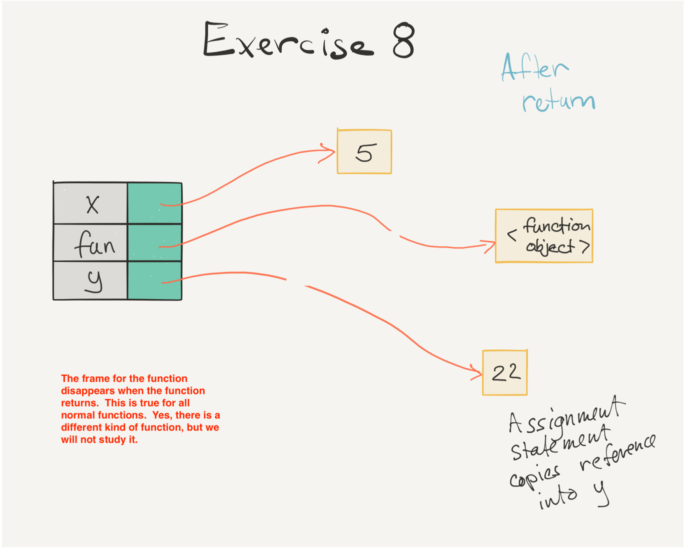
Example 8

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
x = 5

def fun(a, b):
    x = a + b
    return x * 2

y = fun(x, x + 1)
```


Exercise 8
Before function call
x
fun
y
5
< function object >


Exercise 8
Before return
x
fun
y
a
b
x
5
< function object >
6
11
22
This is the value to be returned.



The frame for the function disappears when the function returns. This is true for all normal functions. Yes, there is a different kind of function, but we will not study it.

Example 9

Draw a diagram that shows the frames, variables, values and references for the following Python code:

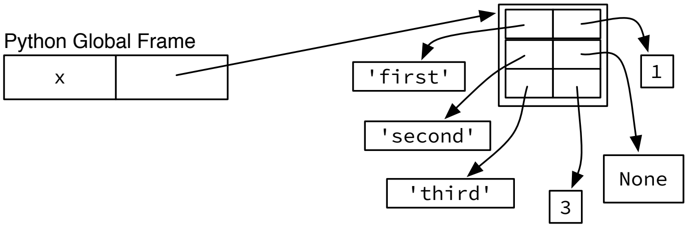
```python
x = [1, 2]

def fun2(a, b):
    a.append(b)
    a.append(b)
    return a

y = fun2(x, x[0])
```

Example 10

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```python
x = {'first': 1, 'second': None}
x['third'] = 3
```

In the diagram, a dictionary is a rectangle with a table inside. The left column stores references to keys, and the right column stores references to values.

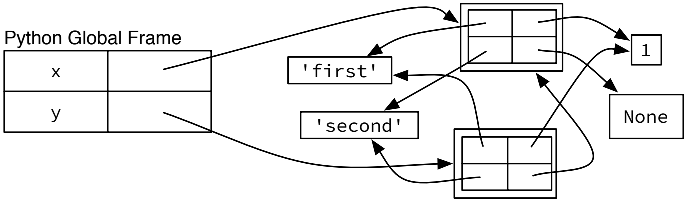


Example 11

Draw a diagram that shows the frames, variables, values and references for the following Python code:

```
x = {'first': 1, 'second': None}
y = {'first': 2, 'second': x}
```

The second line of code creates a new dictionary, with some of the same keys and values. But the second dictionary refers to the first!



Example 12

1. Find a small integer that Python caches; prove it is not created more than once.
2. Find an integer that Python does not cache; prove that it is not cached.
3. Show that floating point values are not cached.
4. Are any other kinds of data values cached? How could you check?

The value 0 is cached.

```python
x = 0
y = 5 - 5
print('Equal value:', x == y, '.  Same object:', x is y)
```

On my Mac, the value 10000 is not part of the cache for small integers.

```python
x = 10000
y = 20000 // 2
print('Equal value:', x == y, '.  Same object:', x is y)
```

Try this!

```
>>> x = 0.0
>>> y = 1.0 - 1.0
>>> x == y
True
>>> x is y
False
```

String literals are cached when the interpreter reads the script.

```
>>> x = 'string'
>>> y = 'string'
>>> z = 'a small string'
>>> z = z[8:]
>>> x == y
True
>>> x is y
True
>>> x == z
True
>>> x is z
False
```