Contents

Introduction

Lab materials adapted from the UC Berkely Data 6

The objectives for this lab are to:

- learn more about Python including:
  - call expressions
  - names and assignment statements
  - Jupyter Memory Model
  - `str` Data Type
  - the `print()` function

Part 1

Call Expressions

Call Expressions in Python apply a function to some arguments.

Also called function calls.

```
max(2, 3)               # Evaluates to 3
max(4, min(1, 9))       # Evaluates to 4
-abs(max(4, 5, -1))     # Evaluates to -5
```

To evaluate a a function call in Python:

1. Evaluate the operands (what is passed into the function), to determine the arguments.
    a. In the first example, the arguments to max are 2 and 3.
    b. In the second example, the arguments to max are 4 and 1.
2. Call the function on the arguments.

Nested Evaluation

We can nest as many function calls as we want! Evaluate inside-out.

```
min(abs(max(-1, -2, -3, min(4, -2))), max(5, 100))
```

The functions get evaluated from the inside-out.

```
min(abs(max(-1, -2, -3, min(4, -2))), max(5, 100))
min(abs(max(-1, -2, -3, min(4, -2))), 100)
min(abs(max(-1, -2, -3, -2)), 100)
min(abs(-1), 100)
min(1, 100)
1
```

Notes on Functions

We've seen a few functions so far: `abs`, `max`, `min`.

Call Expression Format	Return Value
`abs(arg)`	Return the absolute value of the provided argument.
`max(arg1, arg2, ...)`	Return the maximum value of the provided arguments. Requires at least 2 arguments.
`min(arg1, arg2, ...)`	Return the minimum value of the provided arguments. Requires at least 2 arguments.

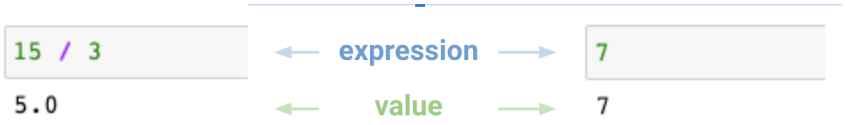
There are many functions built-in to Python, but sometimes we need to use functions that Python doesn't already know about.

- We can use **`import`** statements; we'll learn about this in a few weeks.
- We'll also write our own functions!!! ...in a few weeks.
- Side note: `+`, `-`, `*`, and the other arithmetic operators we saw last lecture are also functions; you'll learn how to implement these in a future class.

Names and Assignment Statements

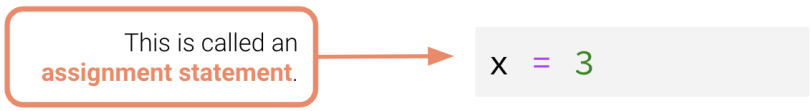
"Storing" Values by Assigning Names

So far we've seen how Python evaluates expressions into values:



A critical aspect of a programming language is its ability to **"store"** and **"retrieve"** values.

In Python, storage and retrieval of values is done by binding names to values with the assignment operator (=).



Assignment Statements

Figure 1: Analogy of Assignment statements to a Luggage Tag

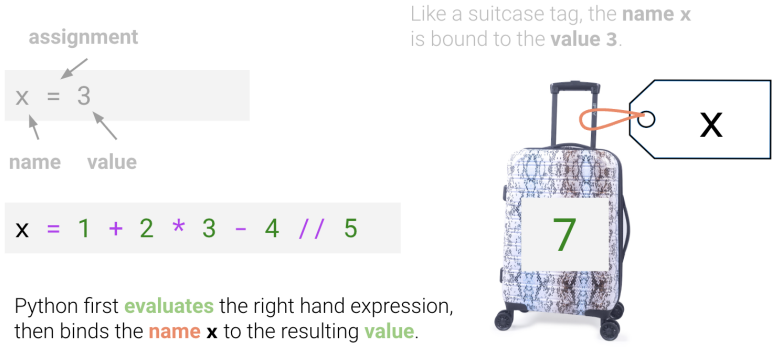


Figure 2: Analogy of Assignment statements, Example 2

Assignment statements do not return anything. They just bind a name to a value.

```
In [34]: x = 3

In [35]: x
Out[35]: 3
```

Why the Suitcase Analogy?

- **Suitcases are containers**, which contain things (values) or more suitcases
  - More on this later
- Just like a luggage tag, **the names of values can change** (I swap out the luggage tag)
  - The value (stuff in the suitcase) is still there, we just have a different way of identifying it
- I can use **multiple names to refer to the same value** (you can have multiple luggage tags on your suitcase)

Demo - Data Types

Demo - Data Types

Back to the CDC MMWR Tuberculosis Data from last lecture (source).

U.S. jurisdiction	No. of TB cases			TB incidence		
	2019	2020	2021	2019	2020	2021
Total	8,900	7,173	7,860	2.71	2.16	2.37
Alabama	87	72	92	1.77	1.43	1.83
...	...	...	...	...	...	...
California	2,111	1,706	1,750	5.35	4.32	4.46
...	...	...	...	...	...	...

```
In [1]: cases_2019 = 2111
        type(cases_2019)
Out[1]: int

In [2]: incidence_2019 = 5.35
        type(incidence_2019)
Out[2]: float
```

The function `type()` will return the type of an expression.

Example - Assignment Statements Are Not Mathematical Equations

This statement errors because Python tries to bind the name **3** to the value **x**.

```
In [36]: 3 = x
  File "<ipython-input-36-97d1cbcfdbdb>", line 1
    3 = x
    ^
SyntaxError: cannot assign to literal
```

Example

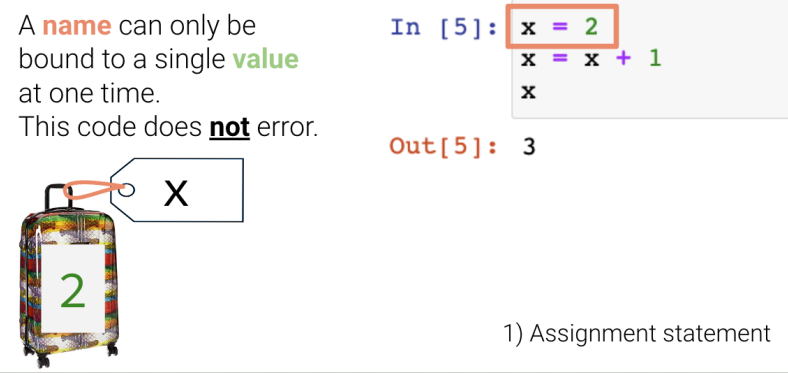


Figure 3: Assignment Statement

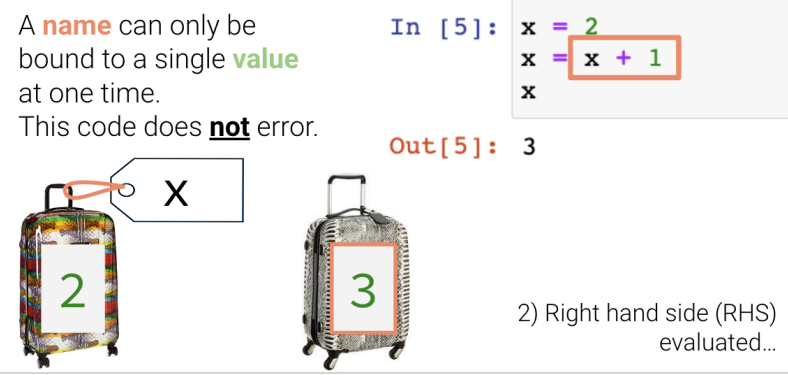


Figure 4: Right Hand Side (RHS) Evaluated

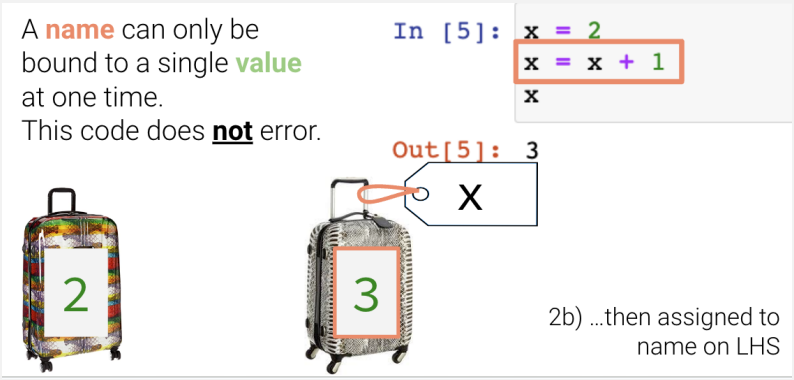


Figure 5: RHS assigned to name on LHS

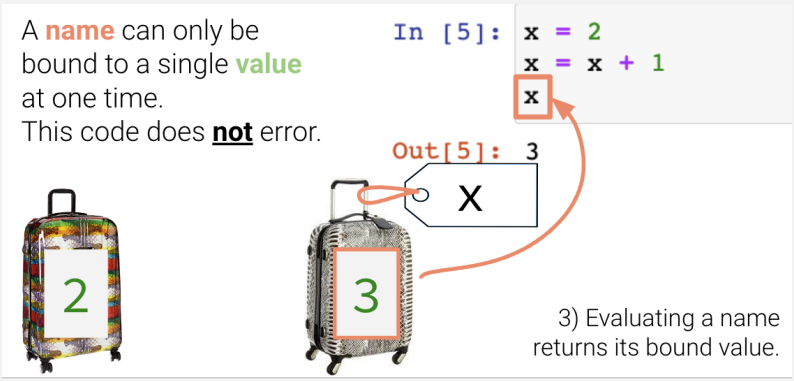


Figure 6: Evaluating a name to return its bound value

Naming

There are some rules on names.

- They **can** contain uppercase and lowercase letters, the digits 0-9, and underscores.
  - Allowed: `how_are_you`, `my_AGE_is_22`, `NETFLIXPASSWORD`
  - Not allowed: `HOW-ARE-YOU`, `ily!`, `AG&*OED`
- They cannot start with a number.
  - Allowed: `my_AGE_is_22`, `_22_is_my_AGE`
  - Not allowed: `22_is_my_AGE`
- They are case sensitive!

- All different: `Dog`, `DOG`, `dog`, `dOG`

Choose names that are precise, but descriptive.

```
# Good
seconds_per_hour = 60 * 60
hours_per_year = 24 * 365
seconds_per_year = seconds_per_hour * hours_per_year

# Not good
i_love_chocolate = 60 * 60 * 24 * 365
```

Quick Check

For each of the code cells below, what is the output?

Confirm with your neighbors

Then, check in the notebook

```
side_length = 5
area = side_length ** 2
side_length = side_length + 2
```

```
side_length
```

```
area
```

Jupyter Memory Model

If you try to do something with an unbound (i.e., unassigned) name, you will get a NameError.

```
In [1]: turtle + 5

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-8503ace1c834> in <module>
----> 1 turtle + 5

NameError: name 'turtle' is not defined
```

Assigning a built-in Python function/keyword as a name will overwrite its actual meaning.
Don't do it!

```
In [49]: max = 9

In [50]: max(2, 3)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-e6f19394c198> in <module>
----> 1 max(2, 3)

TypeError: 'int' object is not callable
```

That being said, **accidents happen**. :) If this happens to you, you should remove that line of code then restart your kernel.

40

Jupyter Memory Model

Pretend your notebook has a brain:

- Everytime you run a cell with an assignment statement, it remembers that name-value binding.
- It will remember all name-value mappings as long as the current **session is open**, no matter how many cells you create.

However! When you open a notebook for the first time in a few hours, your previous session will likely have ended and Jupyter's brain won't remember anything.

You'll need to re-run all of your cells.

Because of the above, once you've bound names to values:

- Don't delete the cell with the assignment statement.
- Don't use names above the cell with the assignment statement.

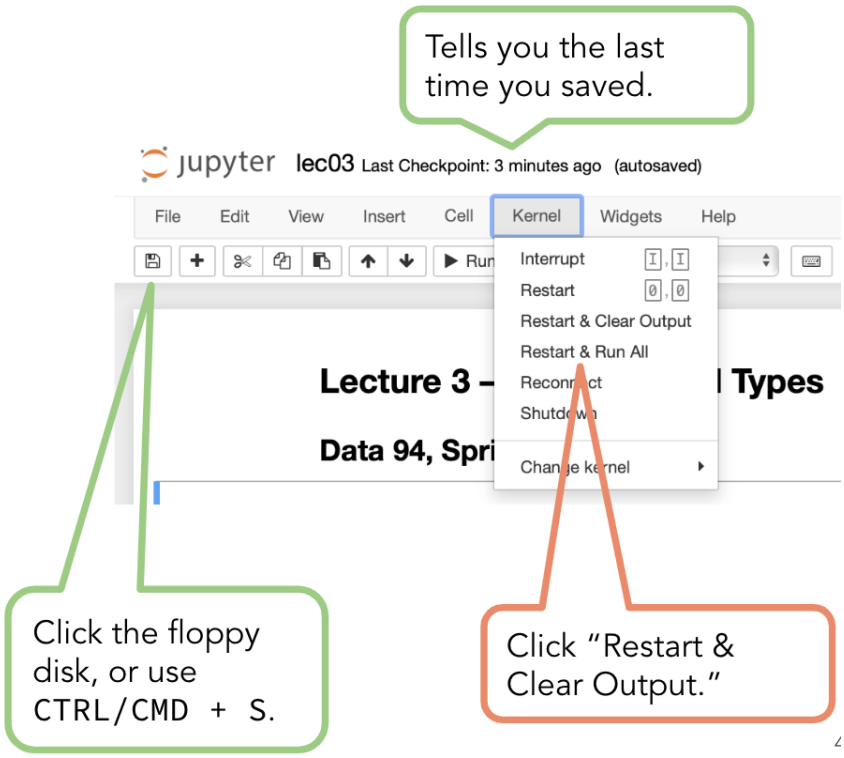
Notebooks should be a paper trail. Each cell is a record of what you have done so far.

How to debug names / restart the kernel

Debug: If you're ever unsure of the value bound to a name, you can simply create a new cell, type the name, and run the cell.

Restart: If something doesn't seem right, always do the following:

1. Save your notebook. You should do this frequently anyways, don't rely on autosave!
2. Restart your kernel.



Strings

Back to the CDC MMWR Tuberculosis Data from last lecture (source).

U.S. jurisdiction	No. of TB cases			TB incidence		
	2019	2020	2021	2019	2020	2021
Total	8,900	7,173	7,860	2.71	2.16	2.37
Alabama	87	72	92	1.77	1.43	1.83
...	...	...	...	...	...	...
California	2,111	1,706	1,750	5.35	4.32	4.46
...	...	...	...	...	...	...

```
In [1]: cases_2019 = 2111
        type(cases_2019)
Out[1]: int

In [2]: incidence_2019 = 5.35
        type(incidence_2019)
Out[2]: float

In [4]: state = "California"
        type(state)
Out[4]: str
```

New type!

The `str` type represents text values.

The function `type()` will return the type of an expression.

Strings

The third key data type in Python is string. Strings are used to store text.

A string is a sequence of characters with single quotes `''` or double quotes `""`.

Since strings are values, they can also be used in assignment statements.

```
"go huskies!!"
'154'
'i am 22 years old'
""
''
```

```
state = "Michigan"
```

A few string operations: * `len()`: string length, i.e., the number of characters in a string

- +: string concatenation

Conversion and Casting

Typecasting

We can also typecast, or convert values between data types.

Note that data type conversions is only valid "when it makes sense"

```
int(x)      # Returns x as an integer
float(x)    # Returns x as a float
str(x)      # Returns x as a string
```

```
In [53]:
# If a float is passed: cuts off decimal
int(3.92)
Out[53]: 3

In [54]:
int(-2.99)
Out[54]: -2

In [55]:
int("-5")
Out[55]: -5

In [56]:
# If a string is passed, it must contain an int
int("4.1")
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-56-bc9758ad056b> in <module>
      1 # If a string is passed, it must contain an int
----> 2 int("4.1")

ValueError: invalid literal for int() with base 10: '4.1'
```

```
In [61]:
str(13 + 14 + 15/2)
Out[61]: '34.5'

In [62]:
str("1")
Out[62]: '1'

In [57]:
float(3)
Out[57]: 3.0

In [58]:
float("3.14159265")
Out[58]: 3.14159265

In [59]:
float(-19.0)
Out[59]: -19.0

In [60]:
float(-14.0 + "3.0")
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-c0fae37873e8> in <module>
----> 1 float(-14.0 + "3.0")

TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

Be Careful!

Different types behave differently when used with the same functions and operators.

```
int, float

3 + 4      # 7
3 + 4.0    # 7.0


3 * 4      # 12
3 * 5.1    # 15.3
3 * 4.0    # 12.0


max(3, 4)      # 4
```

```
str

"3" + "4"      # "34"
"3" + 4      # TypeError!


"3" * 4      # "3333"
"3" * 5.1     # TypeError!
"3" * 4.0     # TypeError!


max("three", "four") # "three"
```

Quick Check

What does the following code output?

Confirm with your neighbors

Then, check in the notebook

```
lucky = 15 + 0.01
lucky = str(int(lucky)) + "3"
int(lucky) - 1
```

`print()`

A very useful function - `print()`

The `print()` function displays values.

- Works even if it's not the last line of a cell!
- Strings are displayed without quotes
- Can take multiple arguments of different types
- Sub-expressions are evaluated before display

```
print(2)
print("Hello, world!")
```

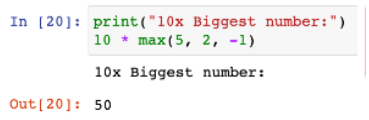
```
2
Hello, world!
```

```
x = 3
y = 4
print(x, "+", y, "is equal to", x + y)
```

```
3 + 4 is equal to 7
```

Note, print displays values.

It does not produce cell output!



Terminology

- **Print** means **display**
- **Output** means the **cell output**

Quick Check

What happens when we run the cell below?

Confirm with your neighbors

Then, check in the notebook

```
print(15)
x = 3 + 4
x
print(14)
```

```
-3
```