EMERITUS
INSTITUTE OF MANAGEMENT

WEEK 3

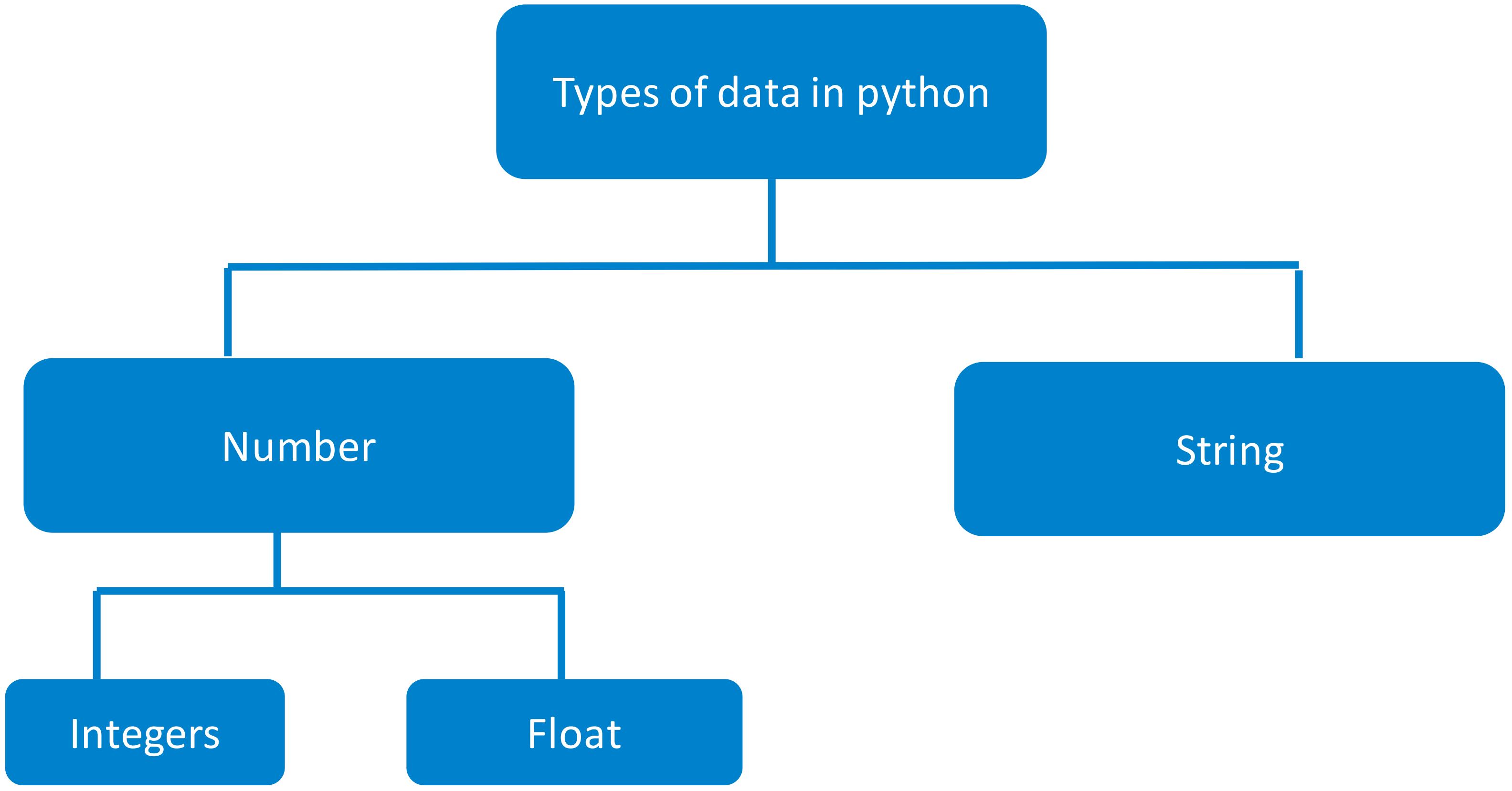
PYTHON BASICS: HOW TO TRANSLATE PROCEDURES INTO CODES

MITSloan Executive Education

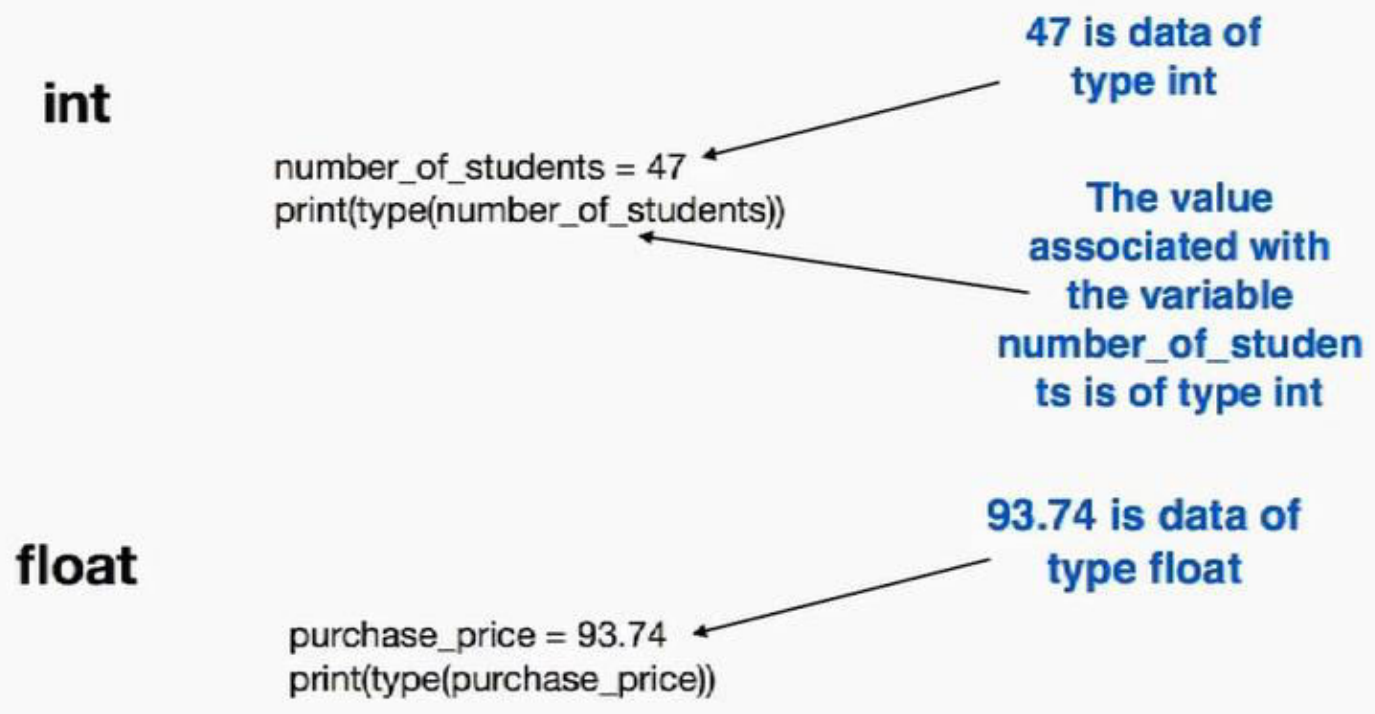
Columbia Business School
AT THE VERY CENTER OF BUSINESS™
EXECUTIVE EDUCATION

Tuck
EXECUTIVE EDUCATION
at Dartmouth

Basic Data Types in Python



Numbers: Integers and Floating



int

```
number_of_students = 47
print(type(number_of_students))
```

47 is data of type int

The value associated with the variable number_of_students is of type int

float

```
purchase_price = 93.74
print(type(purchase_price))
```

93.74 is data of type float

Operations with Numbers

Python notebook

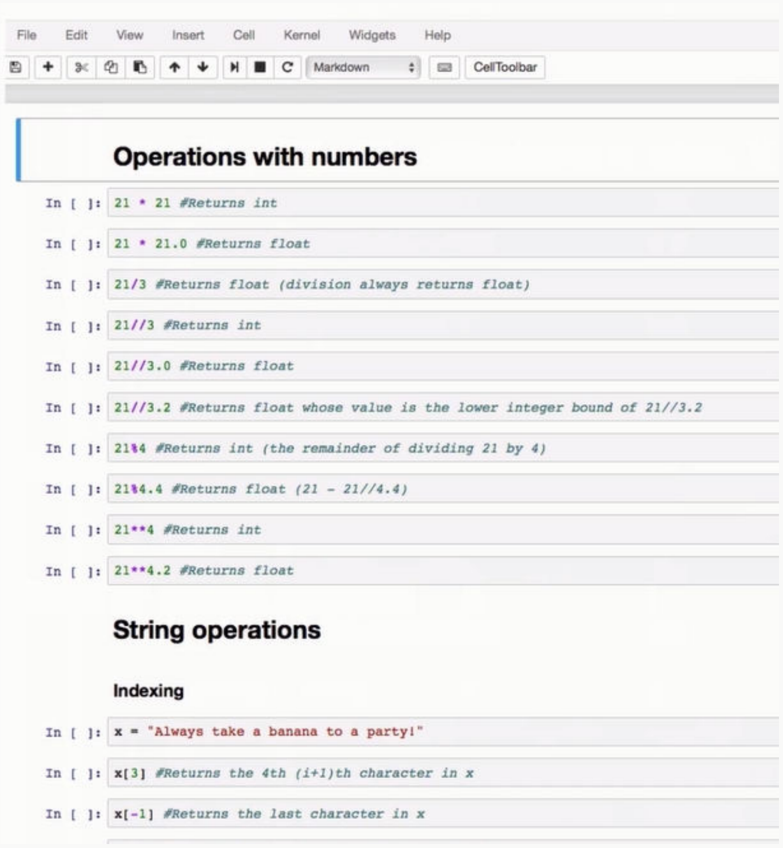
multiplication: x * y

division: x/y
always returns float

integer division: x//y
returns truncated int (as int or float)

remainder: x%y
returns remainder (as int or float)

power: x ** y



Operations with numbers

```
In [ ]: 21 * 21 #Returns int
In [ ]: 21 * 21.0 #Returns float
In [ ]: 21/3 #Returns float (division always returns float)
In [ ]: 21//3 #Returns int
In [ ]: 21//3.0 #Returns float
In [ ]: 21//3.2 #Returns float whose value is the lower integer bound of 21//3.2
In [ ]: 21%4 #Returns int (the remainder of dividing 21 by 4)
In [ ]: 21%4.4 #Returns float (21 - 21//4.4)
In [ ]: 21**4 #Returns int
In [ ]: 21**4.2 #Returns float
```

String operations

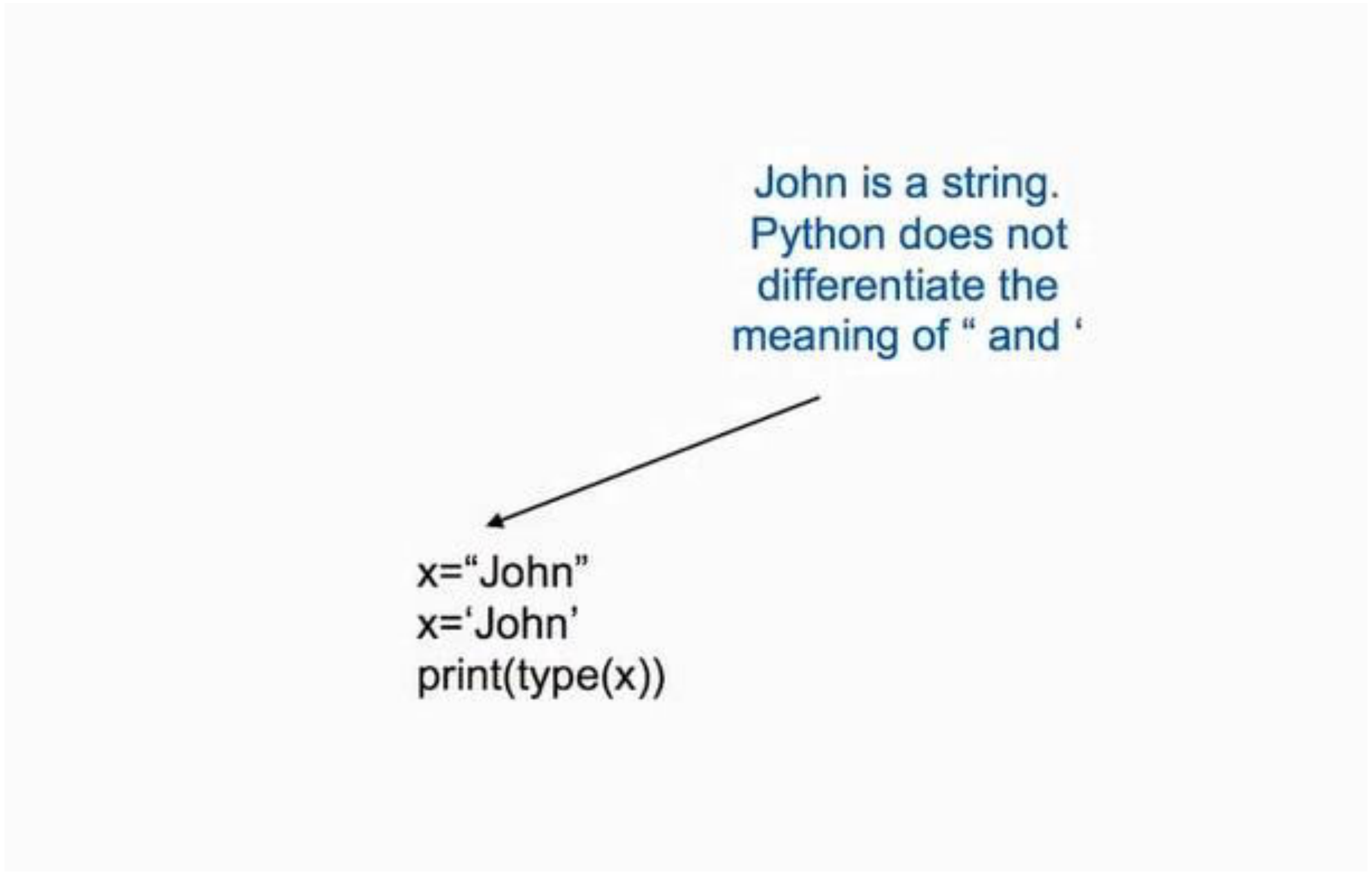
Indexing

```
In [ ]: x = "Always take a banana to a party!"
In [ ]: x[3] #Returns the 4th (i+1)th character in x
In [ ]: x[-1] #Returns the last character in x
```

- Note that if either of the operands in a slash operator are float, then the result is also a float.

Strings

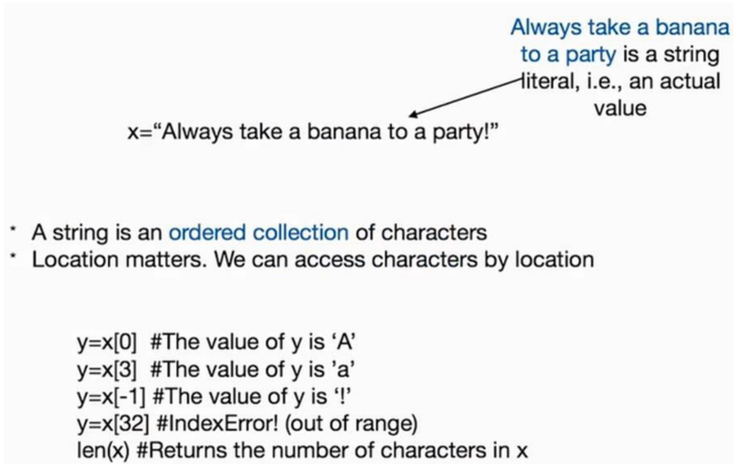
In Python, you can use double quote or single quote to represent string.



John is a string.
Python does not differentiate the meaning of “ and ‘

```
x=“John”
x=‘John’
print(type(x))
```

Strings: Indexing



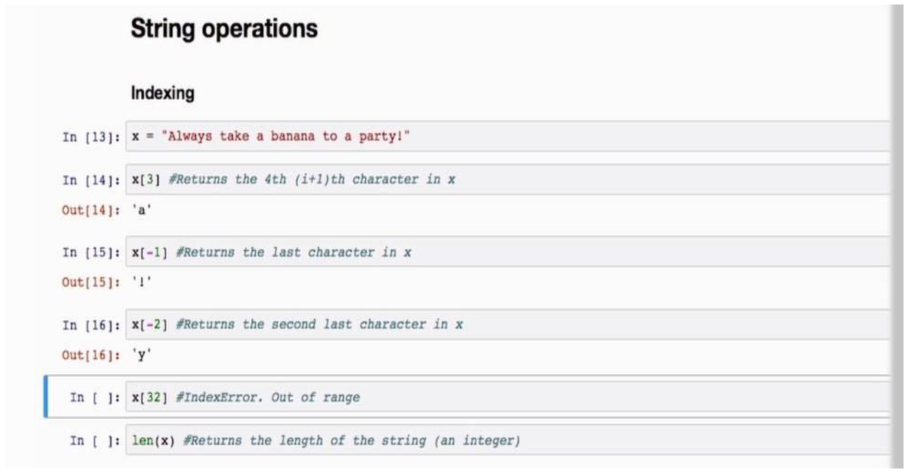
Always take a banana to a party is a string literal, i.e., an actual value

x="Always take a banana to a party!"

- A string is an ordered collection of characters
- Location matters. We can access characters by location

```
y=x[0]  #The value of y is 'A'
y=x[3]  #The value of y is 'a'
y=x[-1] #The value of y is '!'
y=x[32] #IndexError! (out of range)
len(x) #Returns the number of characters in x
```

Python notebook

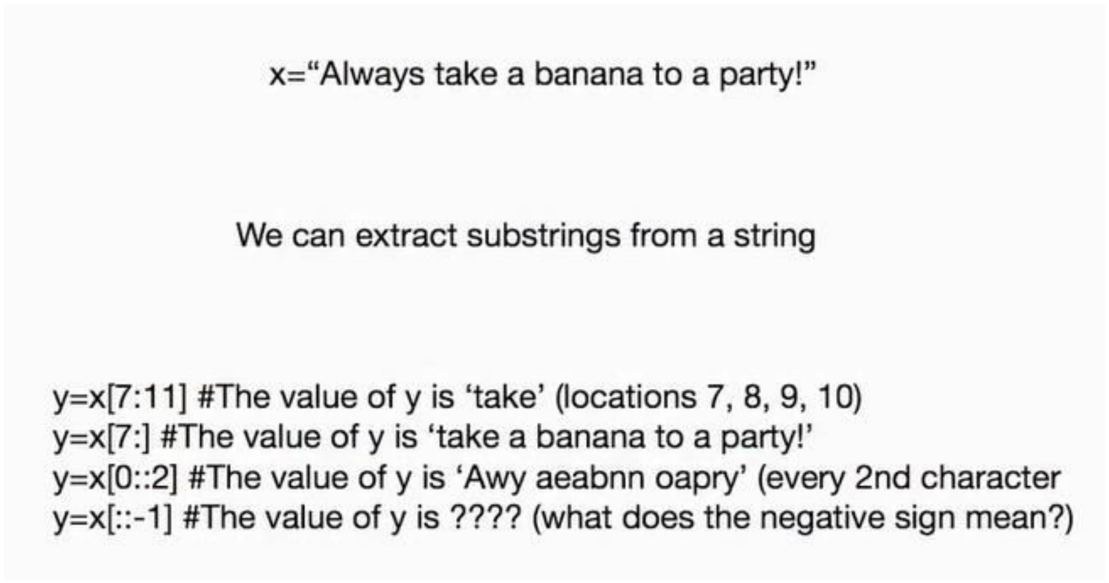


String operations

Indexing

```
In [13]: x = "Always take a banana to a party!"

In [14]: x[3] #Returns the 4th (i+1)th character in x
Out[14]: 'a'

In [15]: x[-1] #Returns the last character in x
Out[15]: '!'

In [16]: x[-2] #Returns the second last character in x
Out[16]: 'y'

In [ ]: x[32] #IndexError. Out of range

In [ ]: len(x) #Returns the length of the string (an integer)
```

Strings: Slicing

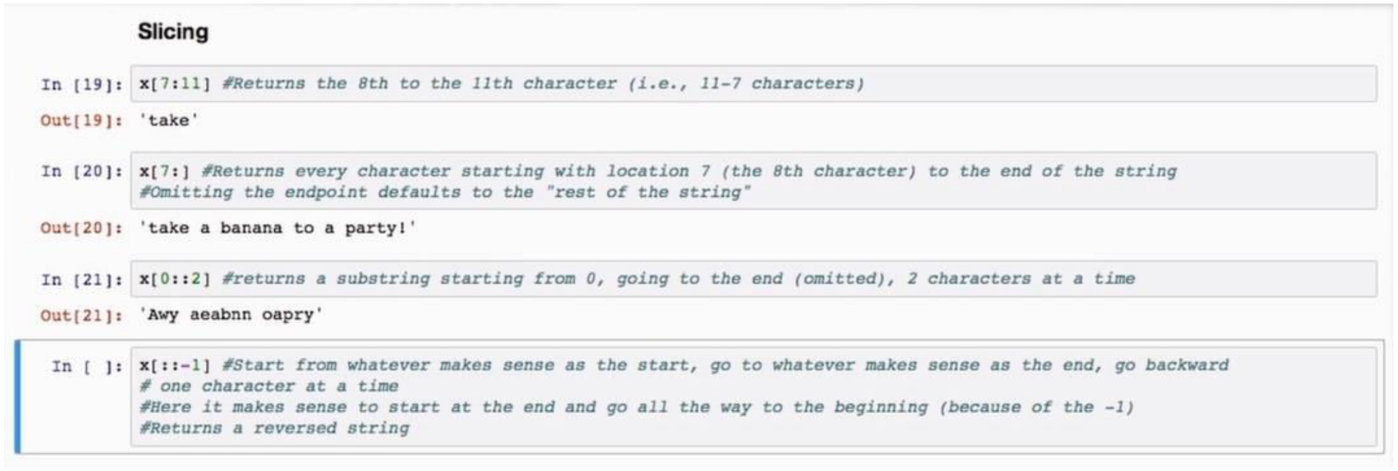


x=“Always take a banana to a party!”

We can extract substrings from a string

y=x[7:11] #The value of y is ‘take’ (locations 7, 8, 9, 10)
y=x[7:] #The value of y is ‘take a banana to a party!’
y=x[0::2] #The value of y is ‘Awy aeabnn oapry’ (every 2nd character
y=x[::-1] #The value of y is ???? (what does the negative sign mean?)

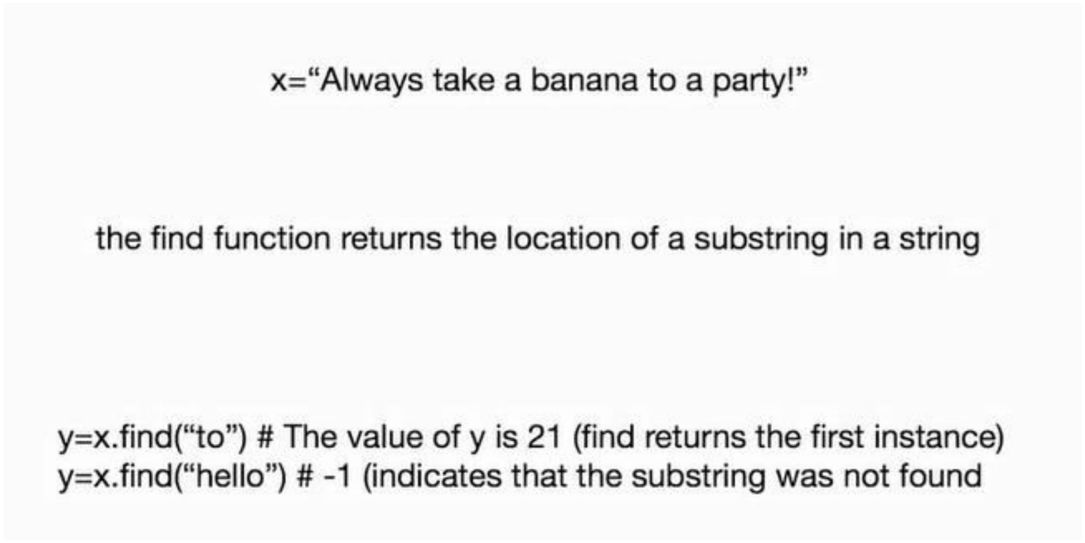
Python notebook



Slicing

```
In [19]: x[7:11] #Returns the 8th to the 11th character (i.e., 11-7 characters)
Out[19]: 'take'

In [20]: x[7:] #Returns every character starting with location 7 (the 8th character) to the end of the string
         #Omitting the endpoint defaults to the "rest of the string"
Out[20]: 'take a banana to a party!'

In [21]: x[0::2] #returns a substring starting from 0, going to the end (omitted), 2 characters at a time
Out[21]: 'Awy aeabnn oapry'

In [ ]:  x[::-1] #Start from whatever makes sense as the start, go to whatever makes sense as the end, go backward
         # one character at a time
         #Here it makes sense to start at the end and go all the way to the beginning (because of the -1)
         #Returns a reversed string
```

Strings: Search

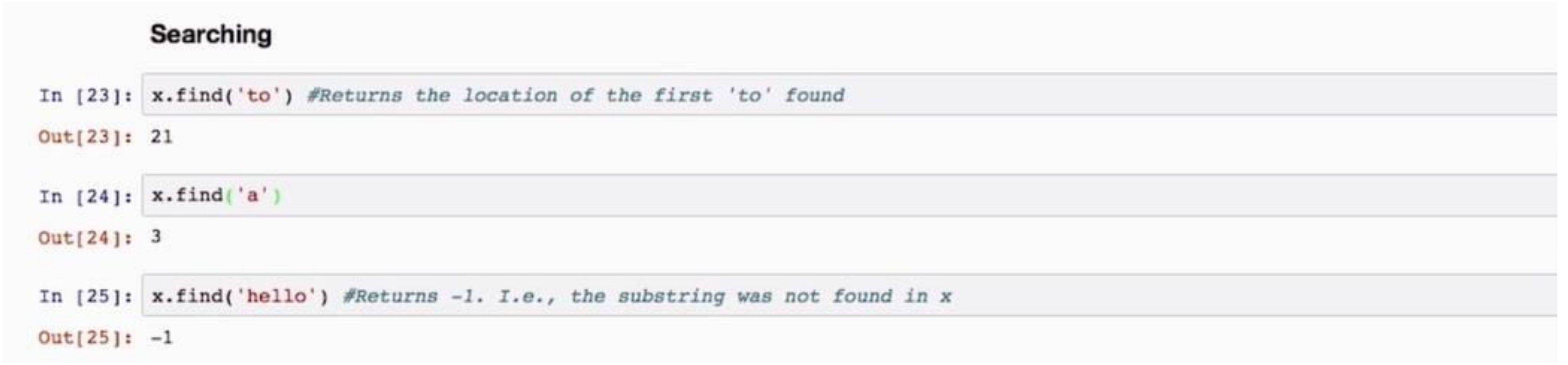


x="Always take a banana to a party!"

the find function returns the location of a substring in a string

y=x.find("to") # The value of y is 21 (find returns the first instance)
y=x.find("hello") # -1 (indicates that the substring was not found

Python notebook



Searching

```
In [23]: x.find('to') #Returns the location of the first 'to' found
Out[23]: 21

In [24]: x.find('a')
Out[24]: 3

In [25]: x.find('hello') #Returns -1. I.e., the substring was not found in x
Out[25]: -1
```

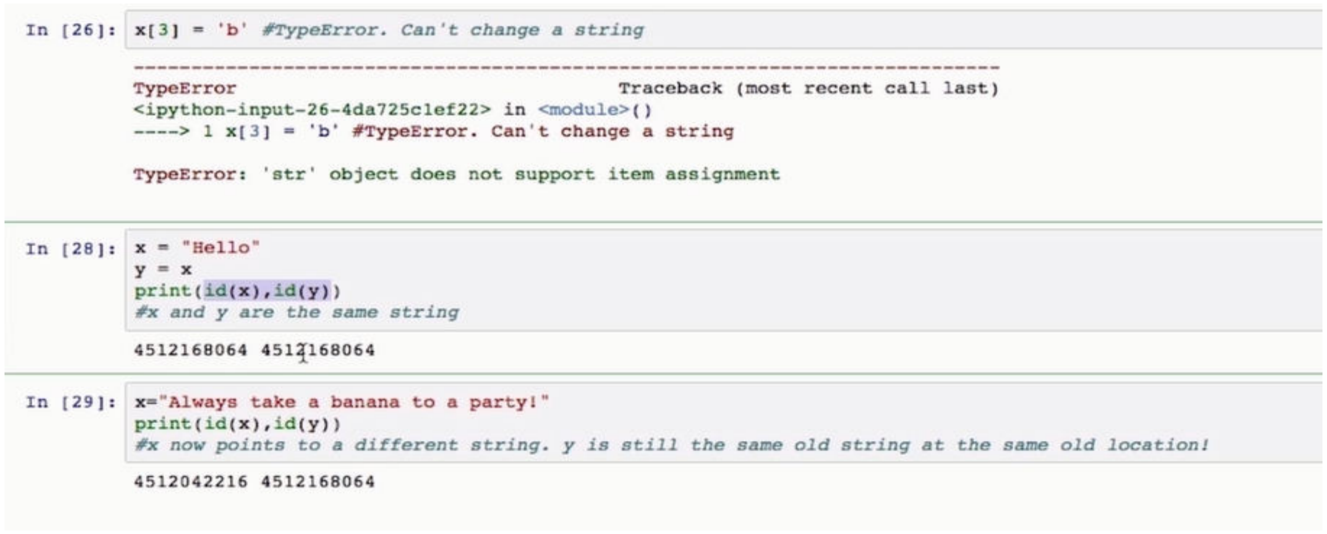
Strings: Immutability

x=“Always take a banana to a party!”

the value of a string cannot be changed

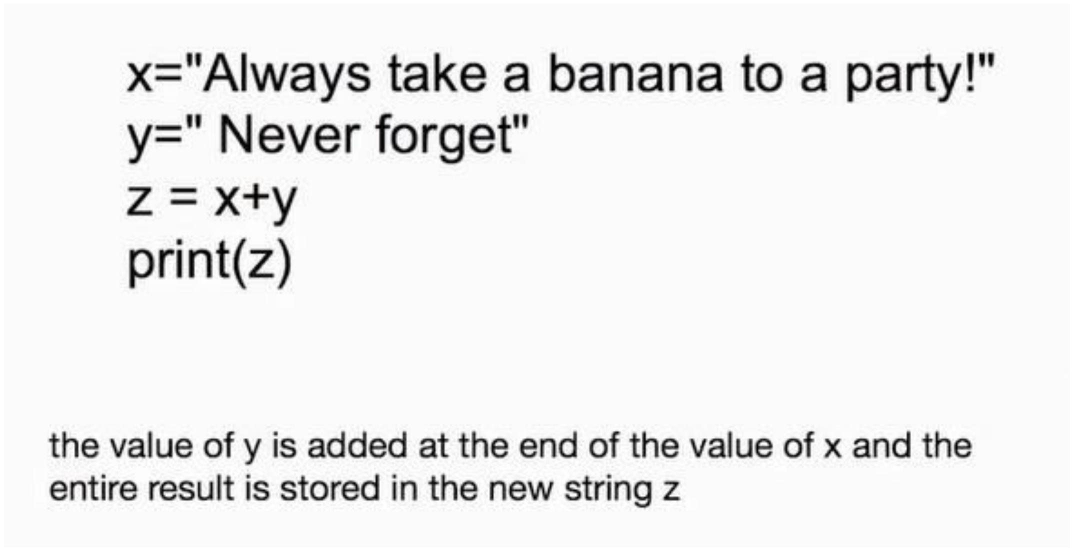
x[5]=‘C’ #TypeError! (string objects are not changeable)

Python notebook



```
In [26]: x[3] = 'b' #TypeError. Can't change a string

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-26-4da725c1ef22> in <module>()
         ----> 1 x[3] = 'b' #TypeError. Can't change a string

         TypeError: 'str' object does not support item assignment

In [28]: x = "Hello"
         y = x
         print(id(x),id(y))
         #x and y are the same string

         4512168064 4512168064

In [29]: x="Always take a banana to a party!"
         print(id(x),id(y))
         #x now points to a different string. y is still the same old string at the same old location!

         4512042216 4512168064
```

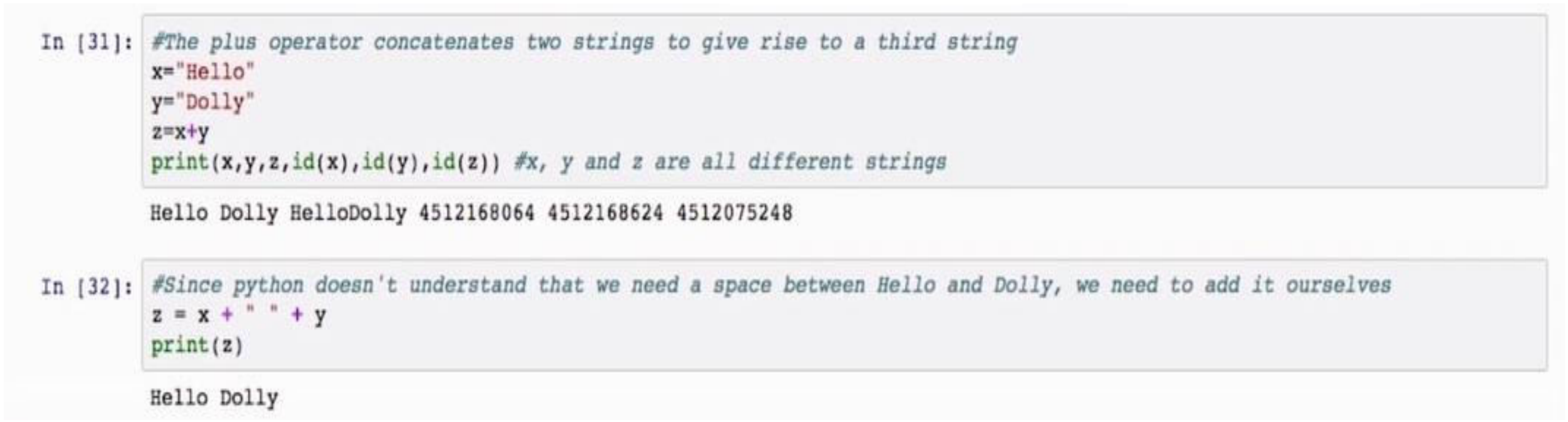
Strings: Concatenation



```
x="Always take a banana to a party!"
y=" Never forget"
z = x+y
print(z)
```

the value of y is added at the end of the value of x and the entire result is stored in the new string z

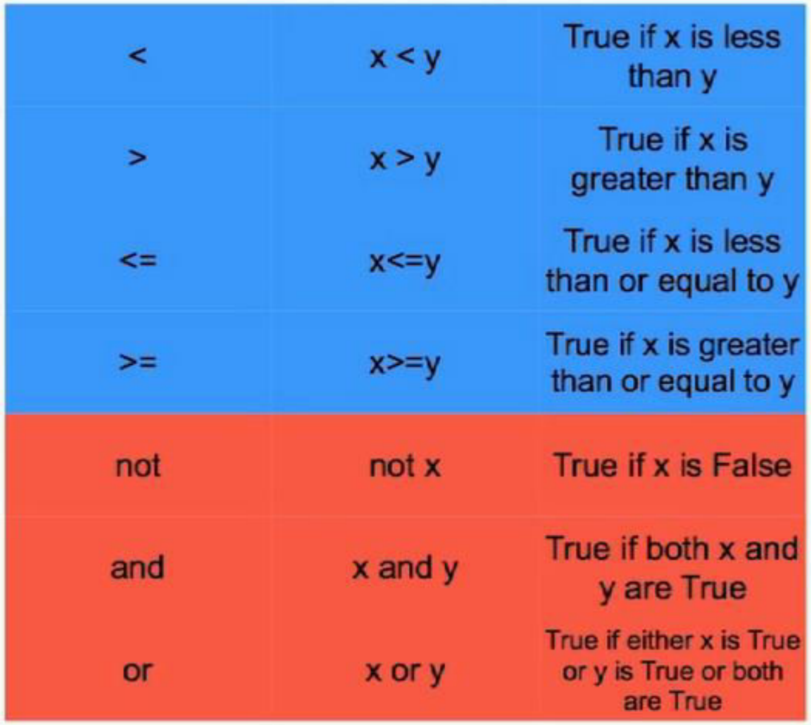
Python notebook



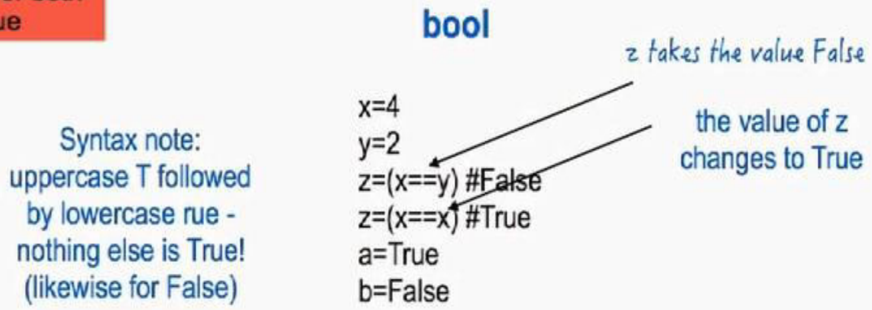
```
In [31]: #The plus operator concatenates two strings to give rise to a third string
         x="Hello"
         y="Dolly"
         z=x+y
         print(x,y,z,id(x),id(y),id(z)) #x, y and z are all different strings

         Hello Dolly HelloDolly 4512168064 4512168624 4512075248

In [32]: #Since python doesn't understand that we need a space between Hello and Dolly, we need to add it ourselves
         z = x + " " + y
         print(z)

         Hello Dolly
```

Strings: Boolean

Relational and Logical Operators



<	x < y	True if x is less than y
>	x > y	True if x is greater than y
<=	x<=y	True if x is less than or equal to y
>=	x>=y	True if x is greater than or equal to y
not	not x	True if x is False
and	x and y	True if both x and y are True
or	x or y	True if either x is True or y is True or both are True



bool

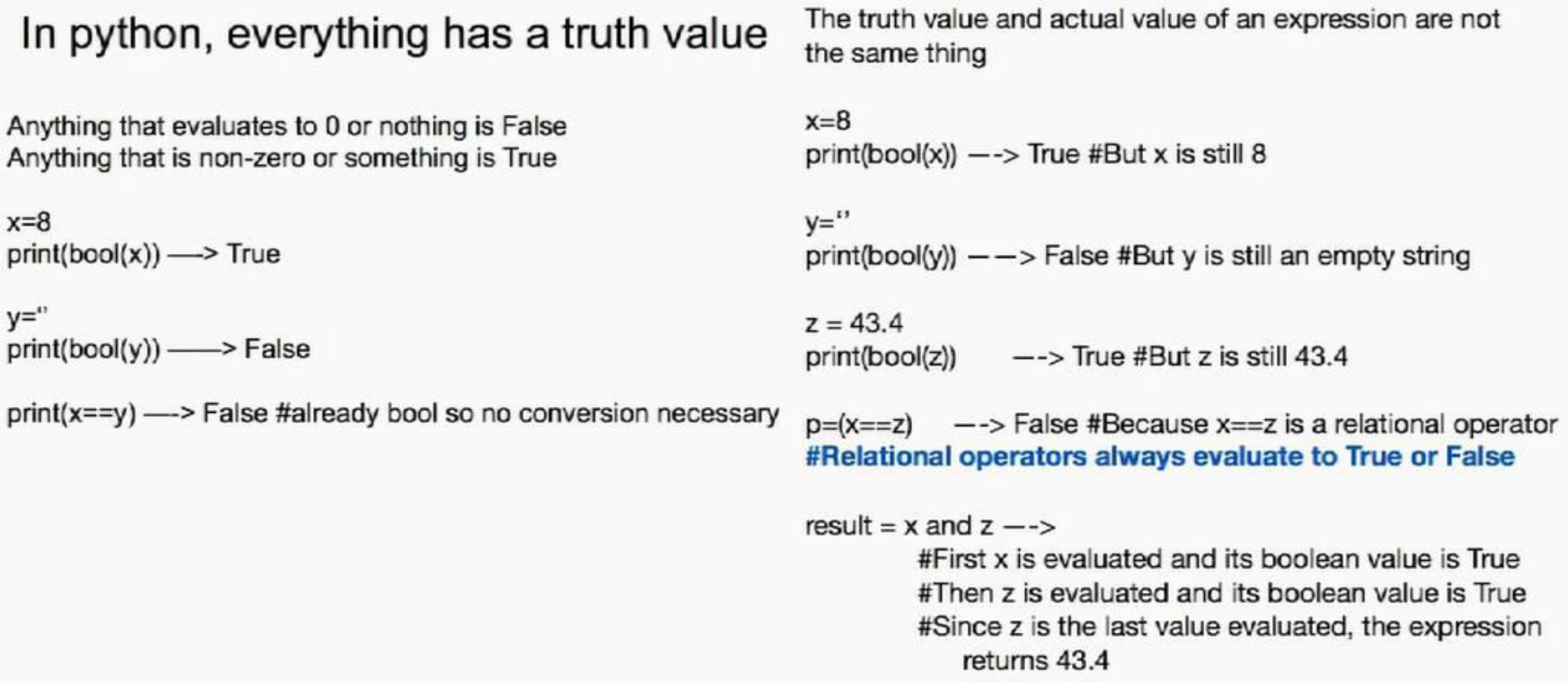
Syntax note: uppercase T followed by lowercase rue - nothing else is True! (likewise for False)

```
x=4
y=2
z=(x==y) #False
z=(x==x) #True
a=True
b=False
```

z takes the value False

the value of z changes to True

Strings: Boolean



In python, everything has a truth value

Anything that evaluates to 0 or nothing is False
Anything that is non-zero or something is True

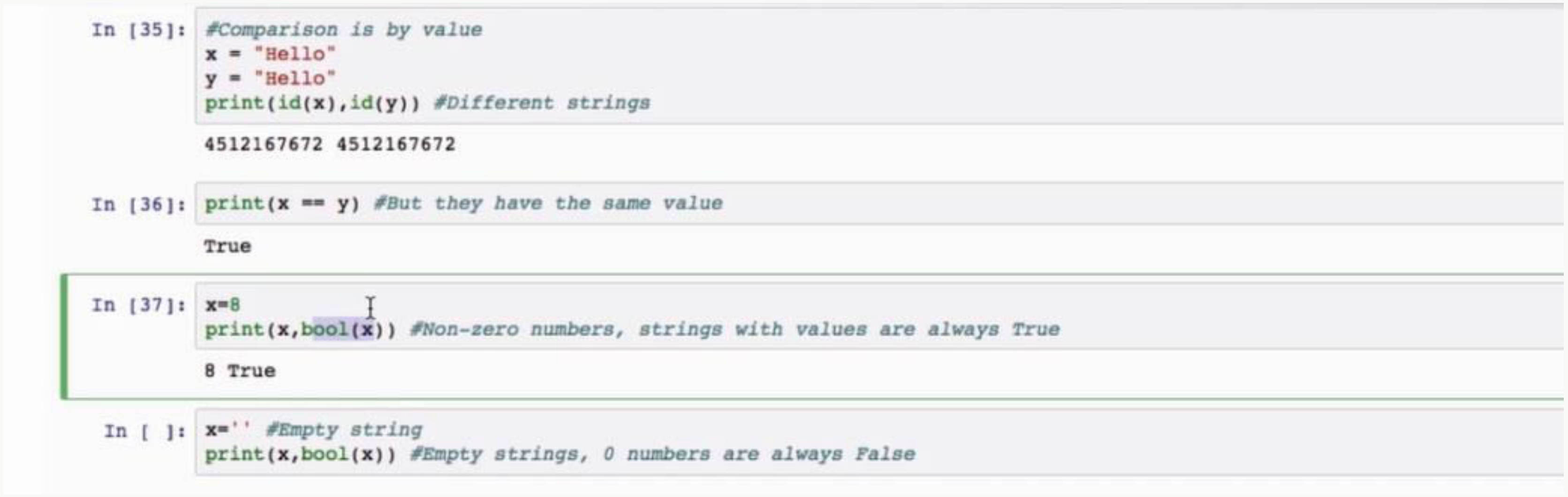
```
x=8
print(bool(x)) —-> True

y=''
print(bool(y)) ——> False

print(x==y) —-> False #already bool so no conversion necessary
```

The truth value and actual value of an expression are not the same thing

```
x=8
print(bool(x)) —-> True #But x is still 8

y=''
print(bool(y)) ——> False #But y is still an empty string

z = 43.4
print(bool(z))      —-> True #But z is still 43.4

p=(x==z)     —-> False #Because x==z is a relational operator
#Relational operators always evaluate to True or False

result = x and z —->
            #First x is evaluated and its boolean value is True
            #Then z is evaluated and its boolean value is True
            #Since z is the last value evaluated, the expression
                returns 43.4
```

Strings: Boolean

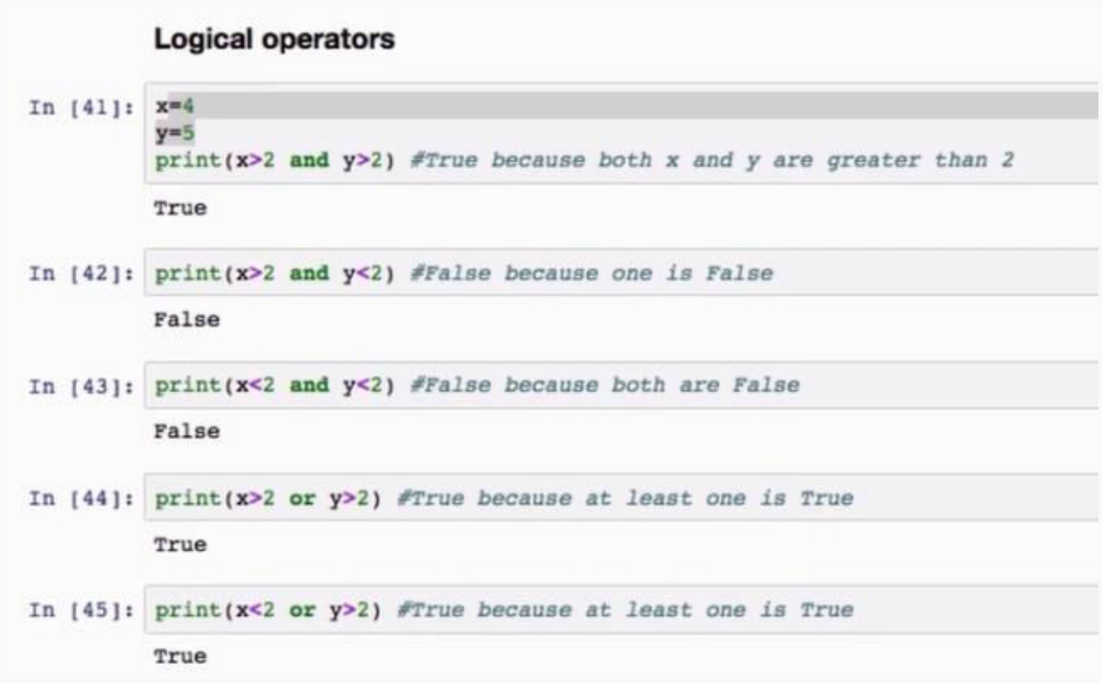
Python notebook



```
In [35]: #Comparison is by value
         x = "Hello"
         y = "Hello"
         print(id(x),id(y)) #Different strings

         4512167672 4512167672

In [36]: print(x == y) #But they have the same value

         True

In [37]: x=8
         print(x,bool(x)) #Non-zero numbers, strings with values are always True

         8 True

In [ ]:  x='' #Empty string
         print(x,bool(x)) #Empty strings, 0 numbers are always False
```

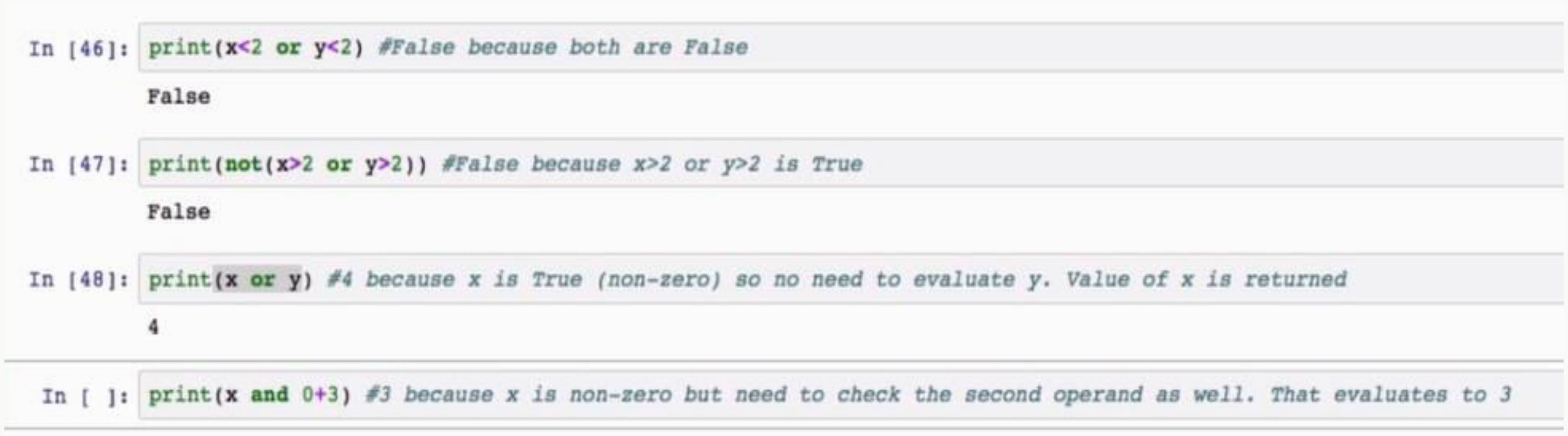
Strings: Logical Operators

Python notebook



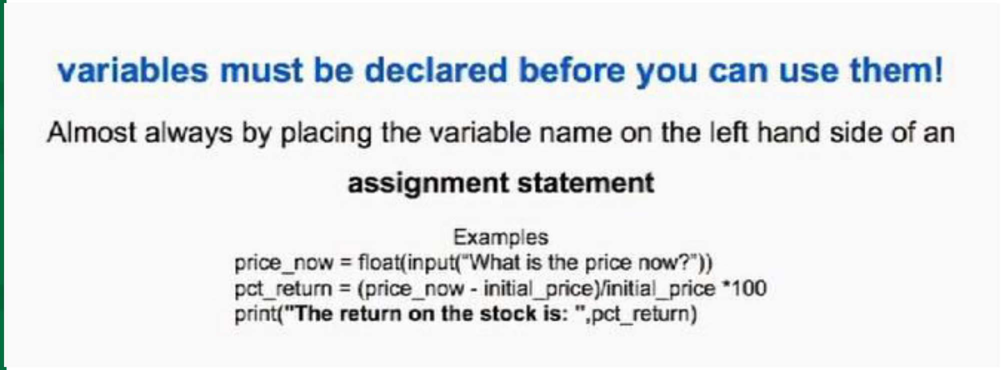
Logical operators

```
In [41]: x=4
         y=5
         print(x>2 and y>2) #True because both x and y are greater than 2

         True

In [42]: print(x>2 and y<2) #False because one is False

         False

In [43]: print(x<2 and y<2) #False because both are False

         False

In [44]: print(x>2 or y>2) #True because at least one is True

         True

In [45]: print(x<2 or y>2) #True because at least one is True

         True
```



```
In [46]: print(x<2 or y<2) #False because both are False

         False

In [47]: print(not(x>2 or y>2)) #False because x>2 or y>2 is True

         False

In [48]: print(x or y) #4 because x is True (non-zero) so no need to evaluate y. Value of x is returned

         4

In [ ]:  print(x and 0+3) #3 because x is non-zero but need to check the second operand as well. That evaluates to 3
```

Variables and Assignments

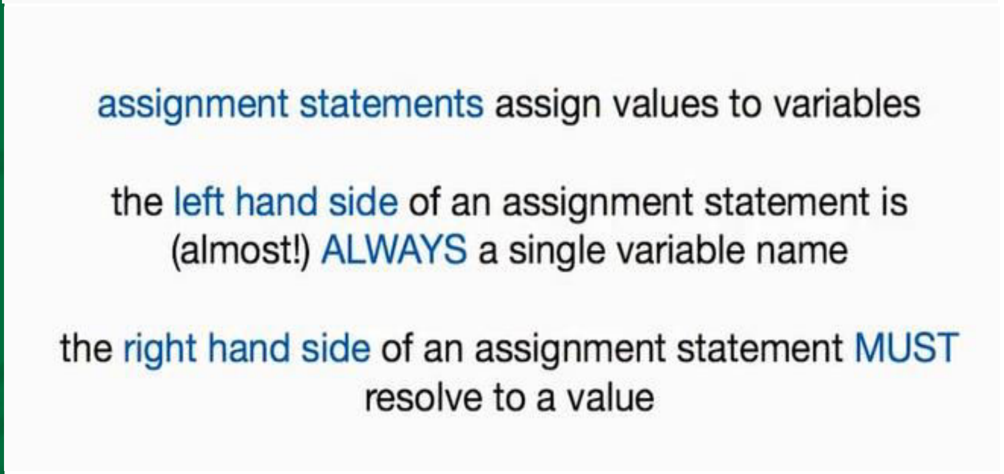


variables must be declared before you can use them!

Almost always by placing the variable name on the left hand side of an **assignment statement**

Examples

```
price_now = float(input("What is the price now?"))
pct_return = (price_now - initial_price)/initial_price *100
print("The return on the stock is: ",pct_return)
```

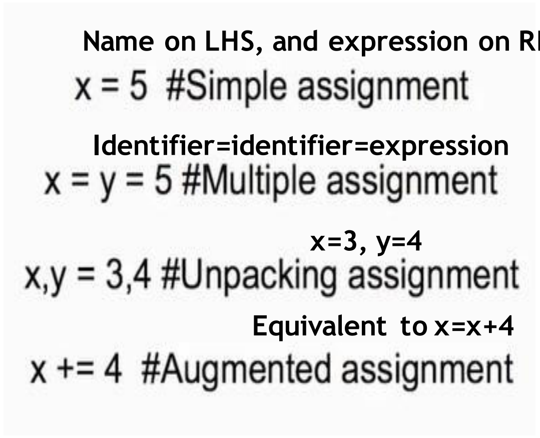


assignment statements assign values to variables

the left hand side of an assignment statement is (almost!) ALWAYS a single variable name

the right hand side of an assignment statement MUST resolve to a value

Types of Assignment Statements



Name on LHS, and expression on RHS

```
x = 5  #Simple assignment
```

Identifier=identifier=expression

```
x = y = 5 #Multiple assignment
```

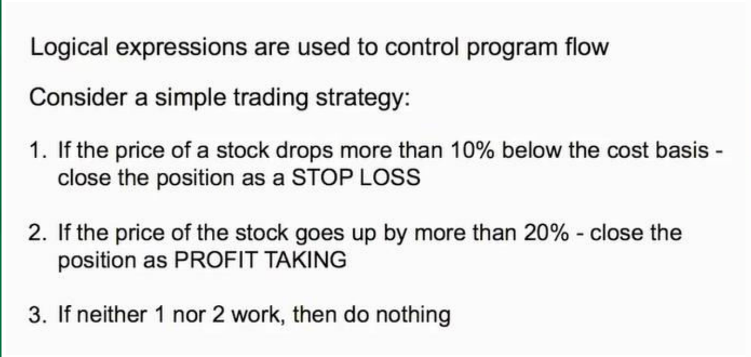
x=3, y=4

```
x,y = 3,4 #Unpacking assignment
```

Equivalent to x=x+4

```
x += 4  #Augmented assignment
```

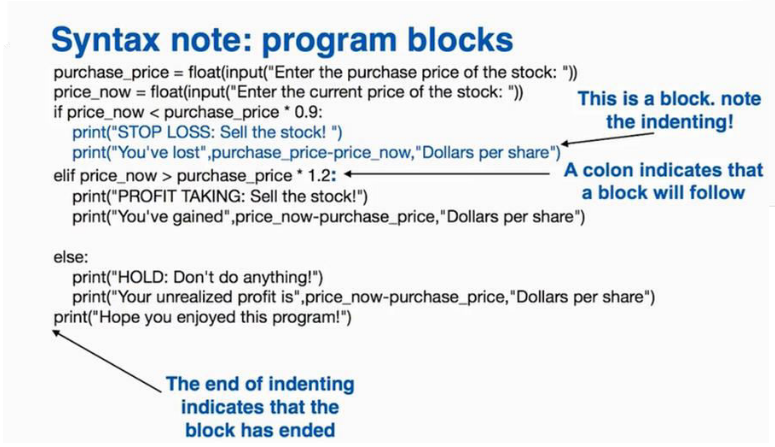
The “if” Statement and Logical Expressions



Logical expressions are used to control program flow

Consider a simple trading strategy:

1. If the price of a stock drops more than 10% below the cost basis - close the position as a STOP LOSS
2. If the price of the stock goes up by more than 20% - close the position as PROFIT TAKING
3. If neither 1 nor 2 work, then do nothing

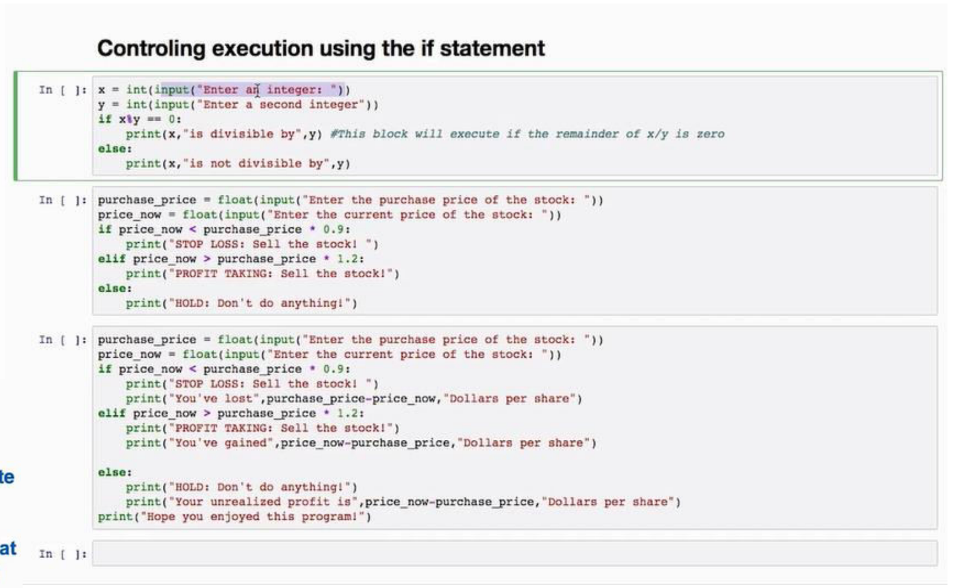


Syntax note: program blocks

```
purchase_price = float(input("Enter the purchase price of the stock: "))
price_now = float(input("Enter the current price of the stock: "))
if price_now < purchase_price * 0.9:
    print("STOP LOSS: Sell the stock! ")
    print("You've lost",purchase_price-price_now,"Dollars per share")
elif price_now > purchase_price * 1.2:
    print("PROFIT TAKING: Sell the stock!")
    print("You've gained",price_now-purchase_price,"Dollars per share")

else:
    print("HOLD: Don't do anything!")
    print("Your unrealized profit is",price_now-purchase_price,"Dollars per share")
print("Hope you enjoyed this program!")
```

This is a block. note the indenting!
A colon indicates that a block will follow
The end of indenting indicates that the block has ended

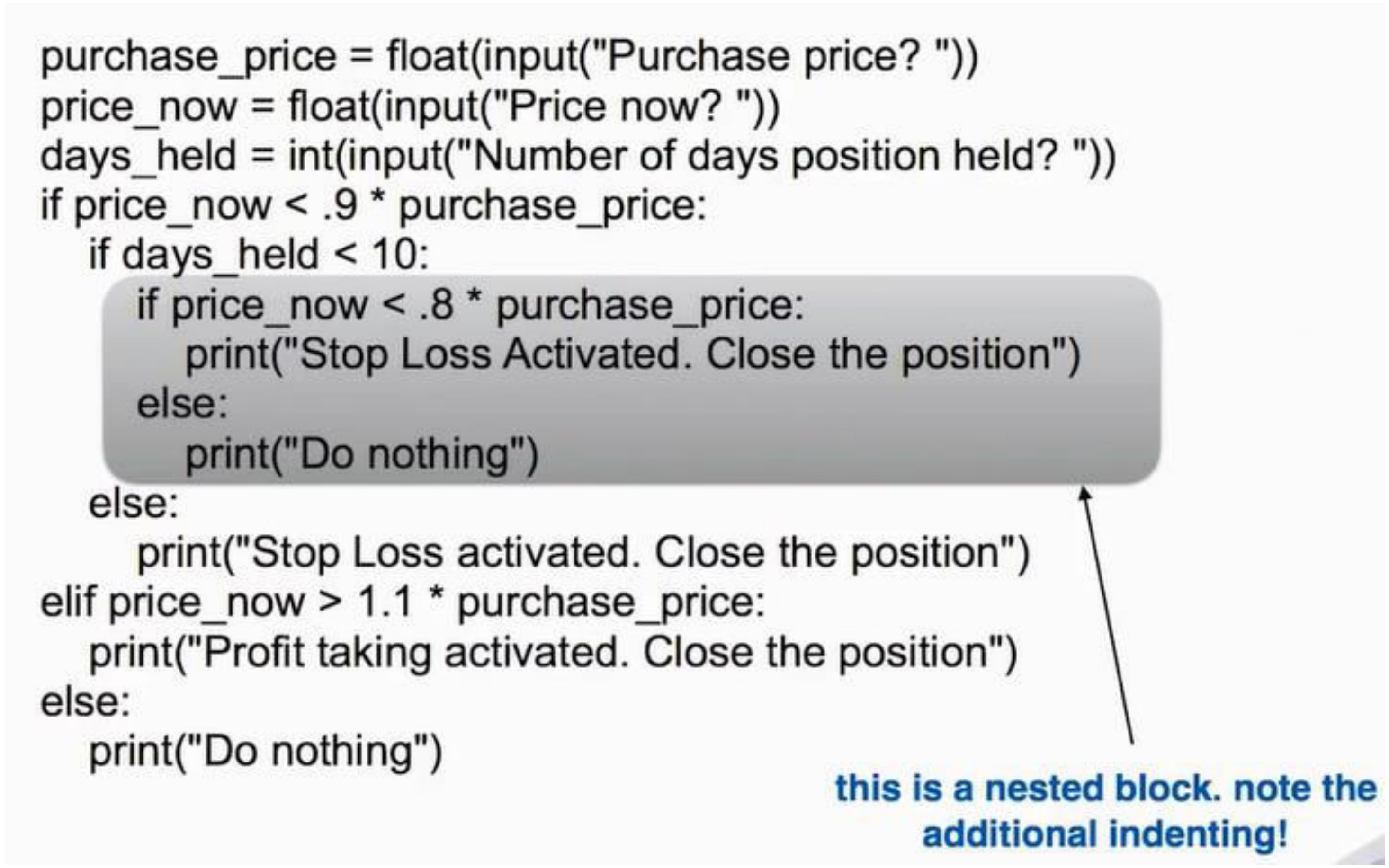
Python notebook



Controling execution using the if statement

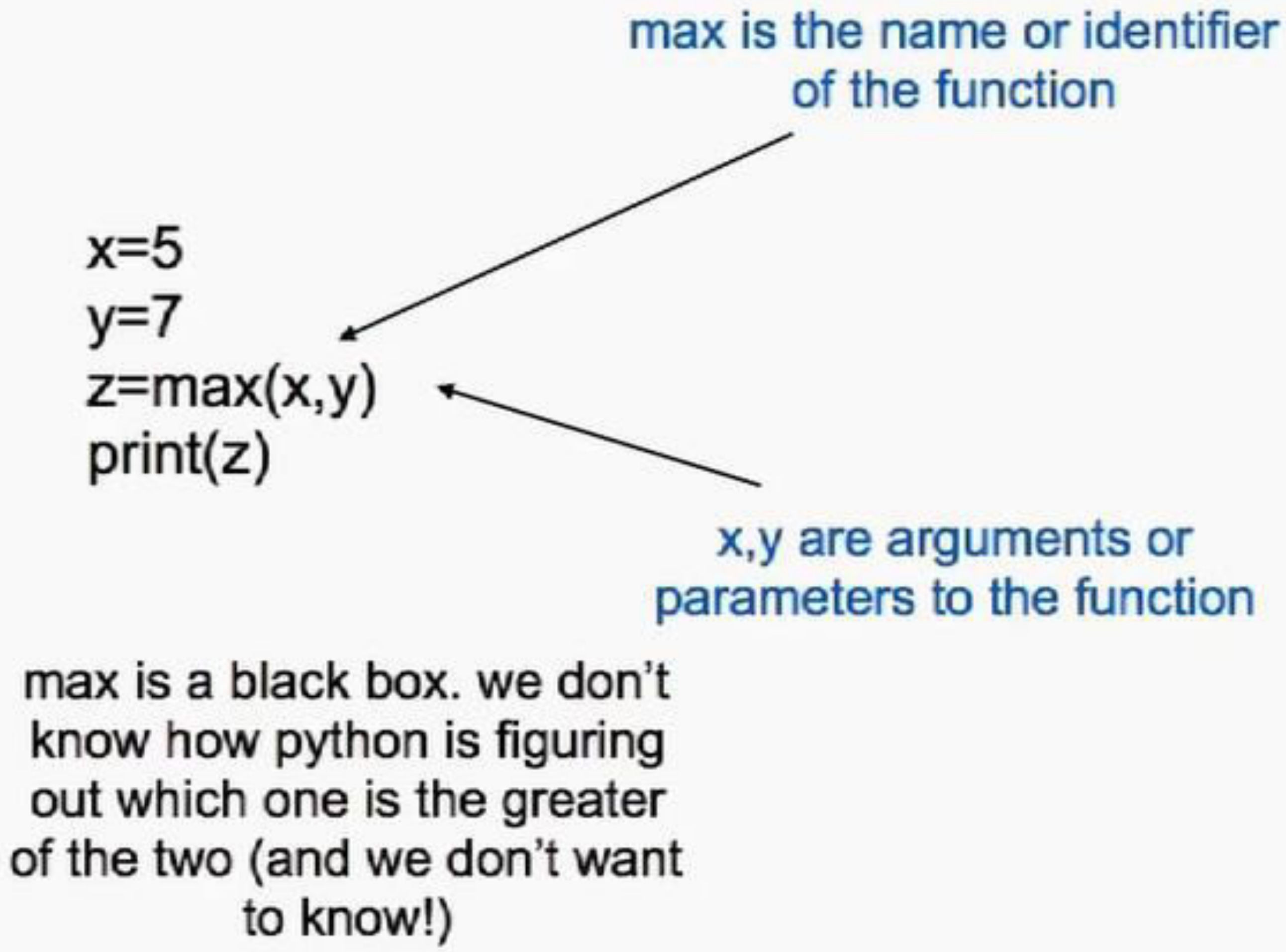
```
In [ ]: x = int(input("Enter an integer: "))
        y = int(input("Enter a second integer"))
        if x%y == 0:
            print(x,"is divisible by",y) #This block will execute if the remainder of x/y is zero
        else:
            print(x,"is not divisible by",y)

In [ ]: purchase_price = float(input("Enter the purchase price of the stock: "))
        price_now = float(input("Enter the current price of the stock: "))
        if price_now < purchase_price * 0.9:
            print("STOP LOSS: Sell the stock! ")
        elif price_now > purchase_price * 1.2:
            print("PROFIT TAKING: Sell the stock!")
        else:
            print("HOLD: Don't do anything!")

In [ ]: purchase_price = float(input("Enter the purchase price of the stock: "))
        price_now = float(input("Enter the current price of the stock: "))
        if price_now < purchase_price * 0.9:
            print("STOP LOSS: Sell the stock! ")
            print("You've lost",purchase_price-price_now,"Dollars per share")
        elif price_now > purchase_price * 1.2:
            print("PROFIT TAKING: Sell the stock!")
            print("You've gained",price_now-purchase_price,"Dollars per share")

        else:
            print("HOLD: Don't do anything!")
            print("Your unrealized profit is",price_now-purchase_price,"Dollars per share")
        print("Hope you enjoyed this program!")

In [ ]:
```

The “if” Statement and Logical Expressions - Nested Blocks



```
purchase_price = float(input("Purchase price? "))
price_now = float(input("Price now? "))
days_held = int(input("Number of days position held? "))
if price_now < .9 * purchase_price:
    if days_held < 10:
        if price_now < .8 * purchase_price:
            print("Stop Loss Activated. Close the position")
        else:
            print("Do nothing")
    else:
        print("Stop Loss activated. Close the position")
elif price_now > 1.1 * purchase_price:
    print("Profit taking activated. Close the position")
else:
    print("Do nothing")
```

Functions: Calling Functions



max is the name or identifier of the function

```
x=5
y=7
z=max(x,y)
print(z)
```

x,y are arguments or parameters to the function

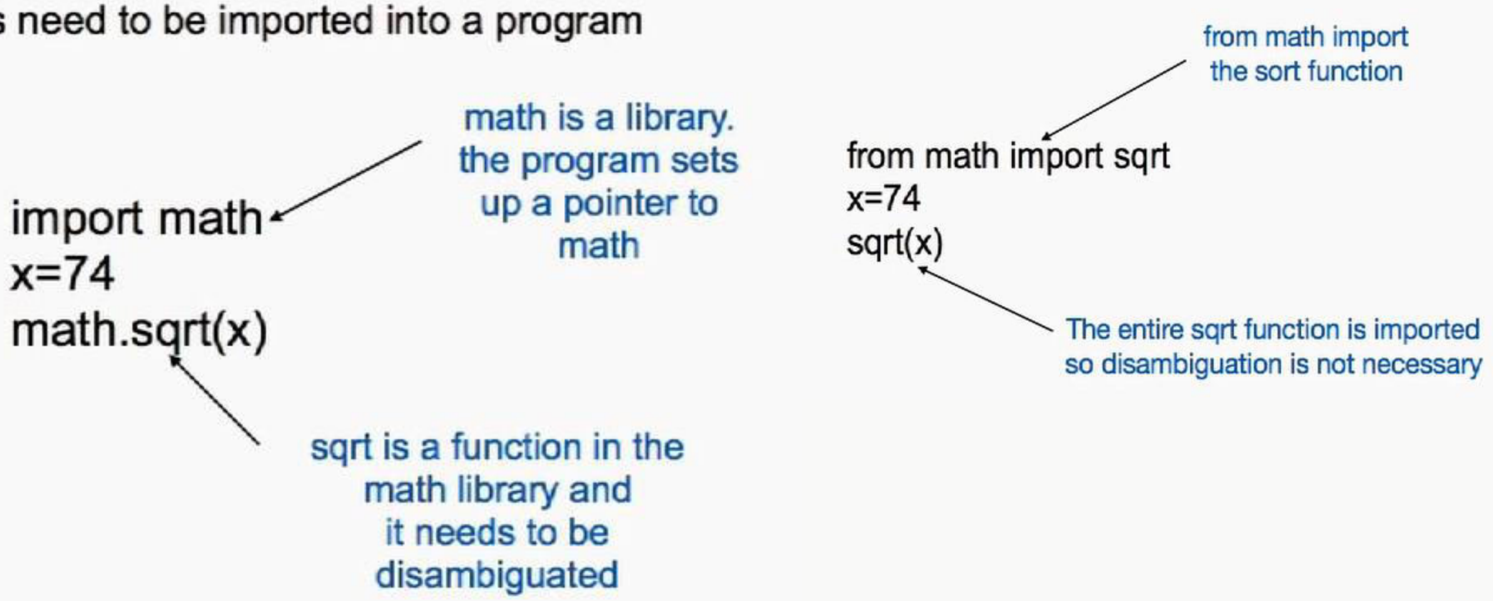
max is a black box. we don’t know how python is figuring out which one is the greater of the two (and we don’t want to know!)

Functions: Function Library

You can also import functions into the library.

Functions can be grouped in libraries

Libraries need to be imported into a program



```
import math
x=74
math.sqrt(x)
```

```
from math import sqrt
x=74
sqrt(x)
```

math is a library. the program sets up a pointer to math
sqrt is a function in the math library and it needs to be disambiguated
from math import the sort function
The entire sqrt function is imported so disambiguation is not necessary

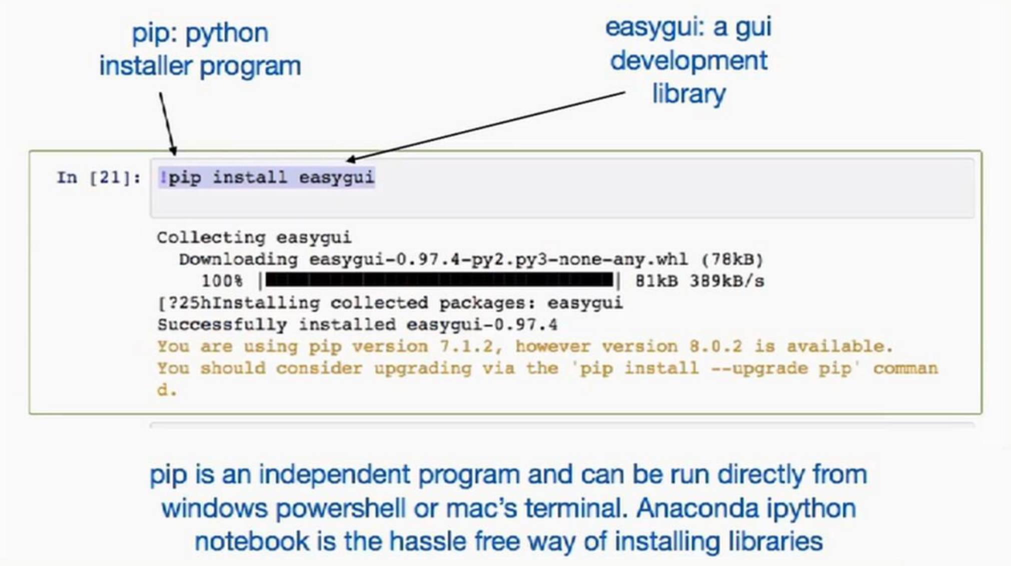
Python is an open source language

With many libraries

Most need to be explicitly installed on your computer

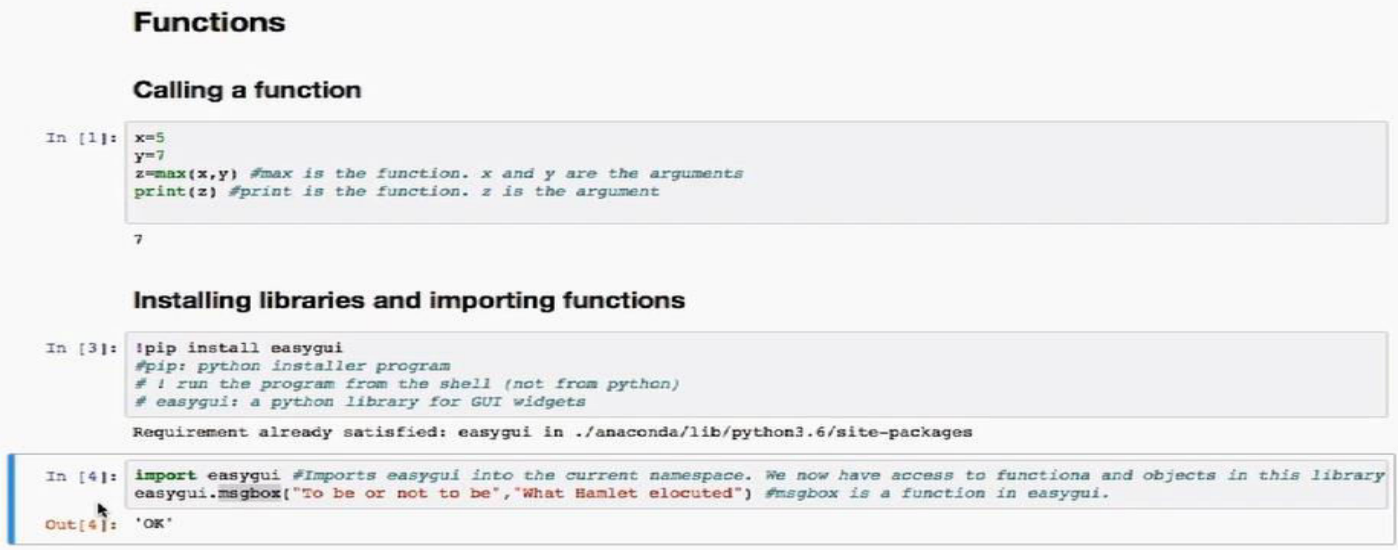
Authenticated libraries are available at https://pypi.python.org/pypi

Functions: Principles of Installing Libraries


pip: python installer program
easygui: a gui development library

```
In [21]: !pip install easygui

Collecting easygui
  Downloading easygui-0.97.4-py2.py3-none-any.whl (78kB)
    100% |████████████████████████████████| 81kB 389kB/s
[?25hInstalling collected packages: easygui
Successfully installed easygui-0.97.4
You are using pip version 7.1.2, however version 8.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' comman
d.
```

pip is an independent program and can be run directly from windows powershell or mac's terminal. Anaconda ipython notebook is the hassle free way of installing libraries



Functions

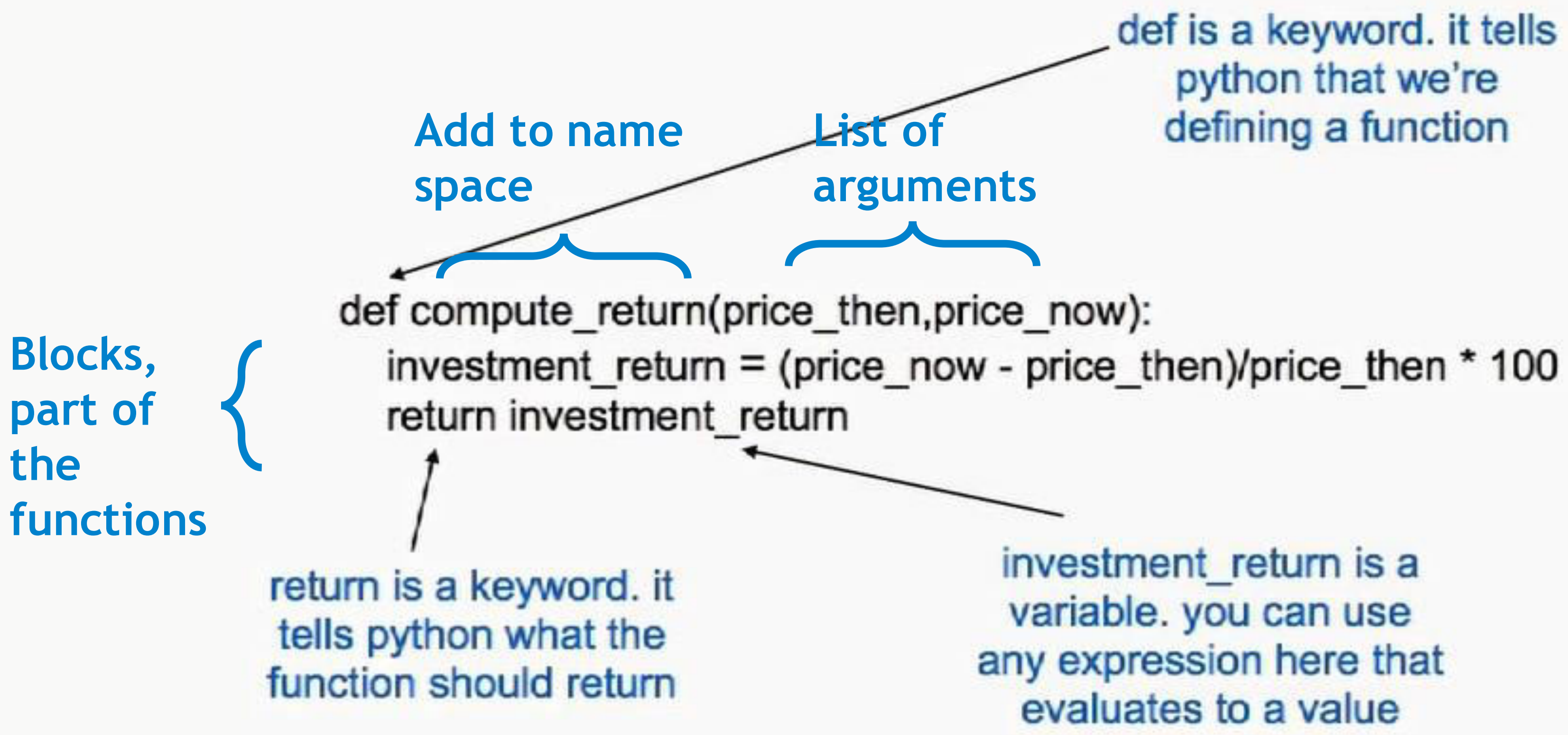
Calling a function

```
In [1]: x=5
        y=7
        z=max(x,y) #max is the function. x and y are the arguments
        print(z) #print is the function. z is the argument

        7
```

Installing libraries and importing functions

```
In [3]: !pip install easygui
        #pip: python installer program
        # ! run the program from the shell (not from python)
        # easygui: a python library for GUI widgets

        Requirement already satisfied: easygui in ./anaconda/lib/python3.6/site-packages

In [4]: import easygui #Imports easygui into the current namespace. We now have access to functiona and objects in this library
        easygui.msgbox("To be or not to be","What Hamlet elocuted") #msgbox is a function in easygui.
Out[4]: 'OK'
```

Functions: Defining your Own Functions



```
def compute_return(price_then,price_now):
    investment_return = (price_now - price_then)/price_then * 100
    return investment_return
```

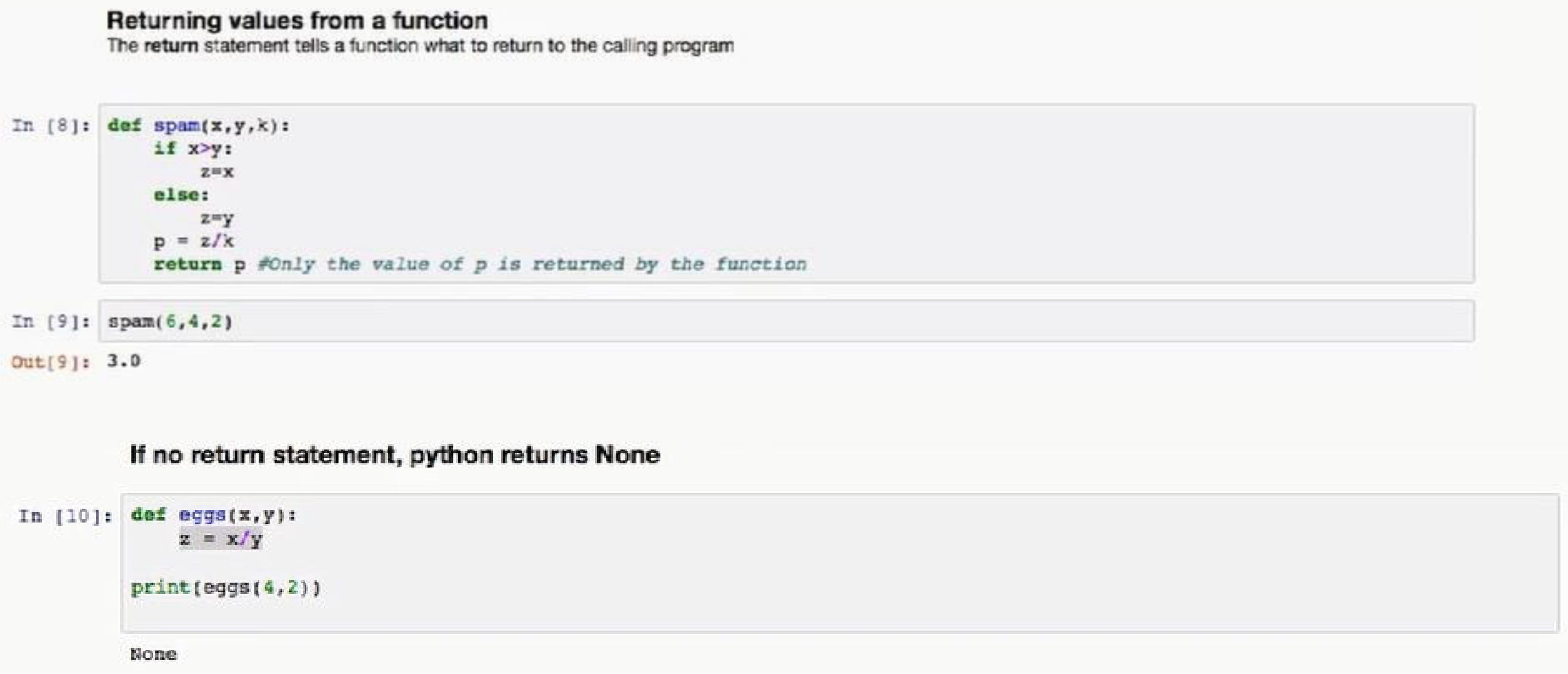
def is a keyword. it tells python that we're defining a function
Add to name space
List of arguments
Blocks, part of the functions
return is a keyword. it tells python what the function should return
investment_return is a variable. you can use any expression here that evaluates to a value

Functions: Return Statement

A function returns a value through the return statement. If there is no return statement, python uses None

```
def spam(x):
  x=x+1

print(spam(5)) —-> None
```

Python notebook



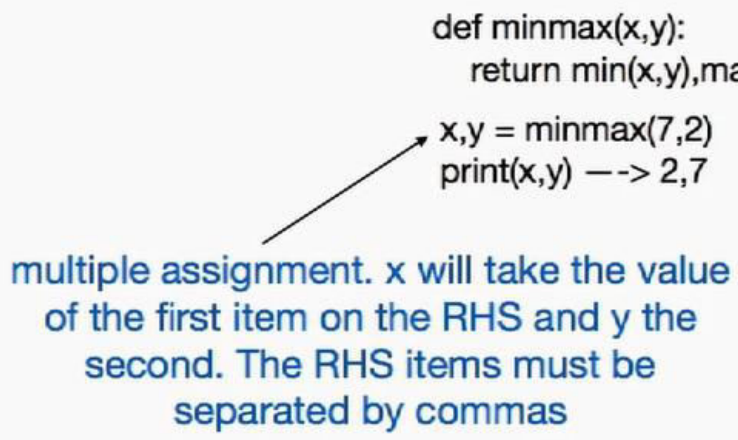
Returning values from a function
The return statement tells a function what to return to the calling program

```
In [8]: def spam(x,y,k):
            if x>y:
                z=x
            else:
                z=y
            p = z/k
            return p #Only the value of p is returned by the function

In [9]: spam(6,4,2)

Out[9]: 3.0
```

If no return statement, python returns None

```
In [10]: def eggs(x,y):
             z = x/y

         print(eggs(4,2))

         None
```

Functions: Returning Multiple Values

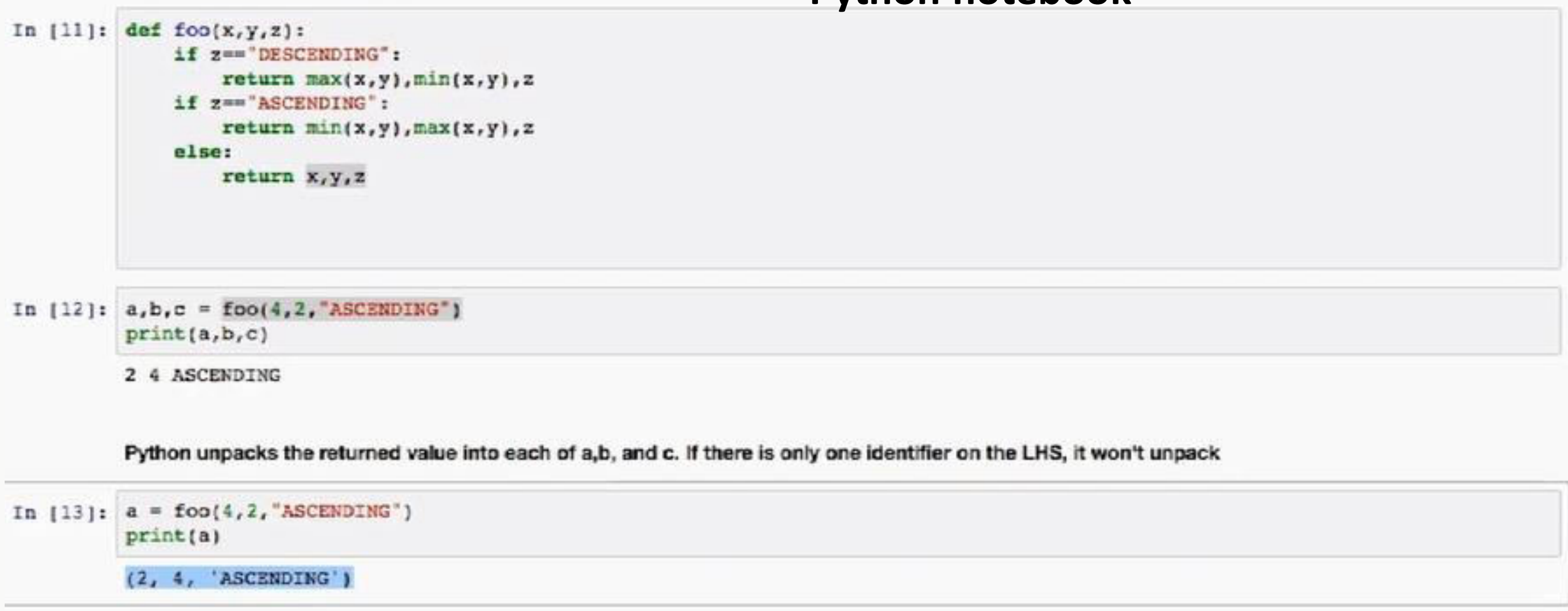


```
def minmax(x,y):
   return min(x,y),max(x,y)

x,y = minmax(7,2)
print(x,y) —-> 2,7
```

multiple assignment. x will take the value of the first item on the RHS and y the second. The RHS items must be separated by commas

#UnpackingAssignment – min(x,y) is assigned to x and max(x,y) is assigned to y

Python notebook



```
In [11]: def foo(x,y,z):
             if z=="DESCENDING":
                 return max(x,y),min(x,y),z
             if z=="ASCENDING":
                 return min(x,y),max(x,y),z
             else:
                 return x,y,z
```

```
In [12]: a,b,c = foo(4,2,"ASCENDING")
         print(a,b,c)

         2 4 ASCENDING
```

Python unpacks the returned value into each of a,b, and c. If there is only one identifier on the LHS, it won't unpack

```
In [13]: a = foo(4,2,"ASCENDING")
         print(a)

         (2, 4, 'ASCENDING')
```

Functions: Passing Arguments to a Function

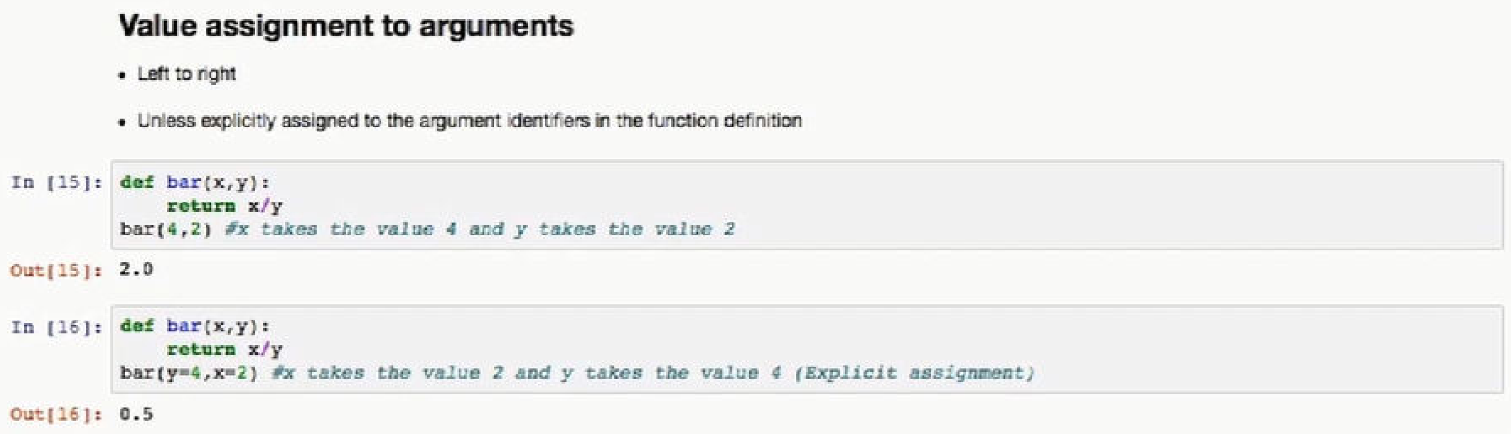
arguments are assigned values from left to right

```
def div(x,y):
  return x/y

a=30
print(div(a,10)) ——> x is 30, y is 10, prints 3
def div(x,y):
  return x/y

x=10
y=30
print(div(y,x)) ——> x is 30, y is 10, prints 3
```

You can give values to arguments directly in a function call

```
def div(x,y):
  return x/y

print(div(x=30,y=10)) —-> 3
print(div(y=10,x=30)) —-> 3
```

Python notebook

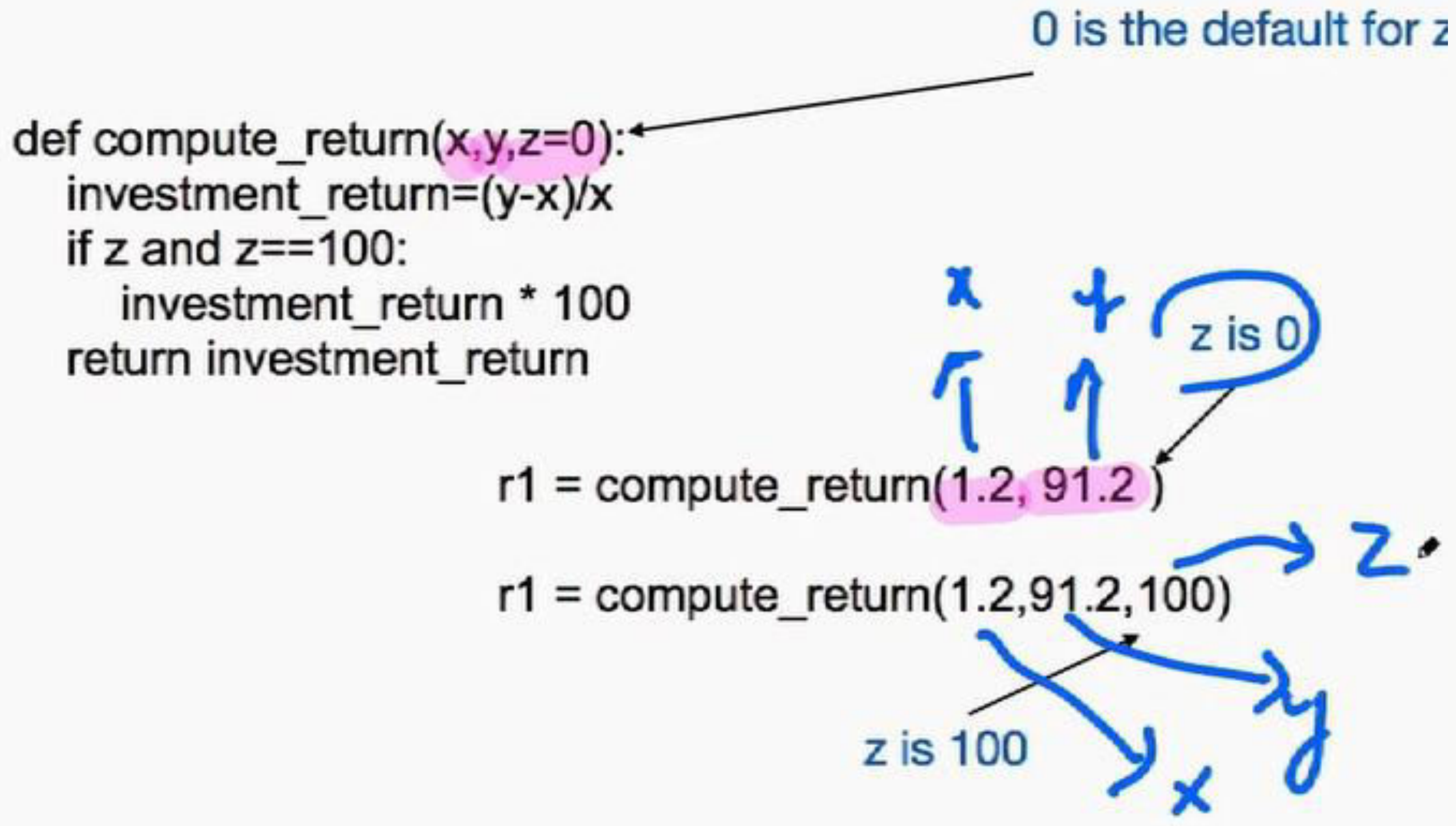


Value assignment to arguments

- Left to right
- Unless explicitly assigned to the argument identifiers in the function definition

```
In [15]: def bar(x,y):
             return x/y
         bar(4,2) #x takes the value 4 and y takes the value 2

Out[15]: 2.0

In [16]: def bar(x,y):
             return x/y
         bar(y=4,x=2) #x takes the value 2 and y takes the value 4 (Explicit assignment)

Out[16]: 0.5
```

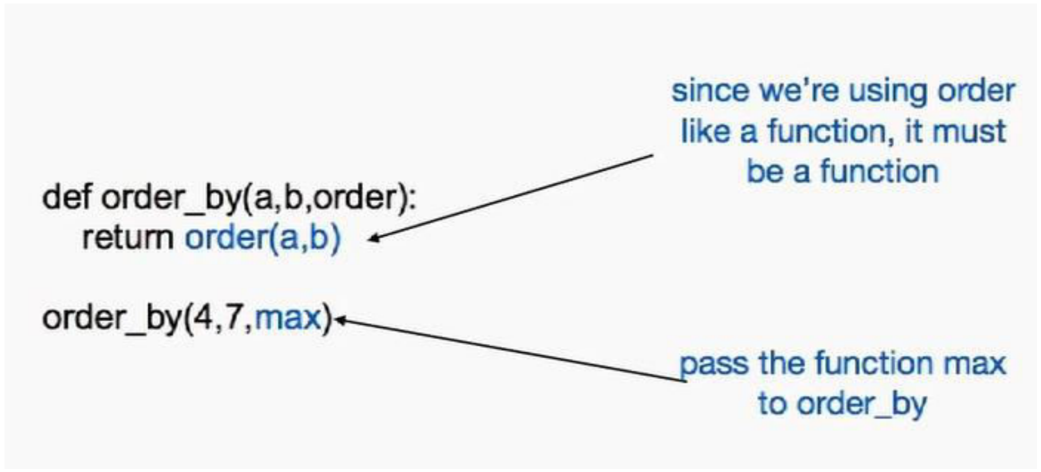
Functions: Default Arguments



```
def compute_return(x,y,z=0):
    investment_return=(y-x)/x
    if z and z==100:
        investment_return * 100
    return investment_return

r1 = compute_return(1.2, 91.2 )

r1 = compute_return(1.2,91.2,100)
```

Functions: Functions as Arguments

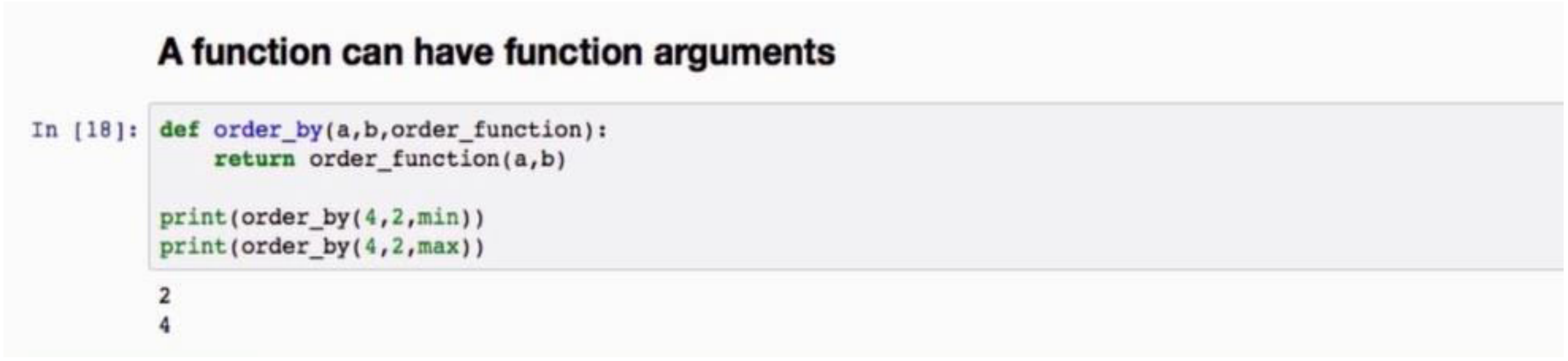
- Functions can have functions as arguments, also called as first order functions in programming languages.
- Python assumes any identifier followed by an open parentheses to be a function.



```
def order_by(a,b,order):
   return order(a,b)

order_by(4,7,max)
```

since we're using order like a function, it must be a function
pass the function max to order_by

Python notebook



A function can have function arguments

```
In [18]: def order_by(a,b,order_function):
             return order_function(a,b)

         print(order_by(4,2,min))
         print(order_by(4,2,max))

         2
         4
```

GLOBAL IVY EDUCATION

www.emeritus.org