Data Analytics for Operations Research

What we will cover

- 1. Essentials of data storage and data access
- 2. Python libraries for data analysis

What this class is not:

- 1. a programming class
- 2. an operations research class

You need to bring:

- 1. competency in programming in any language (we will review python)
- 2. basic knowledge of mathematics and statistics

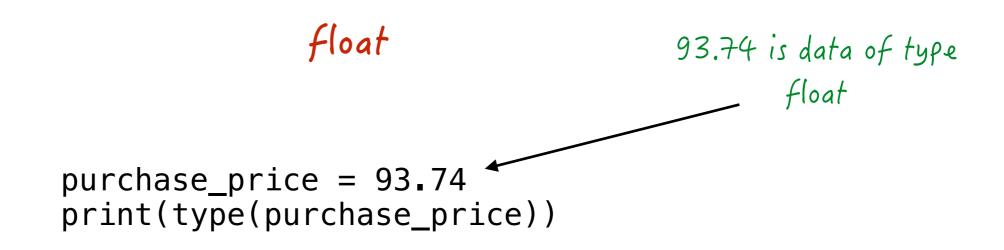
What you should expect

- 1. Lots of assignments
- 2. A couple of quizzes
- 3. An in-class written closed book exam
- 4. A project
- 5. Some amount of class participation

Crash course in Python

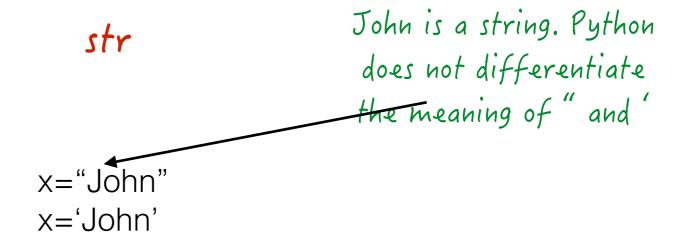
Basic data types in python

number_of_students = 47 print(type(number_of_students)) The value associated with the variable number_of_students is of type int



Operations with numbers

```
x=5
y=3
print(x*y) #multiplication
15
print(x/y) #division
1.6666666666666667
print(x//y) #integer division
1
print(x**y) #raise x to power of y
125
print(x%y) #returns the remainder of x/y
2
```



bool

Syntax note: uppercase

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

z takes the value False

x=4

it's value changes to

y=2

Z=(x==y) #False

z=(x==x) #True

a=True

b=False

Python strings

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

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

```
x is a variable whose value is a string
```

```
y=x[0] #The value of y is 'A'
```

$$y=x[3]$$
 #The value of y is 'a'

$$y=x[7:11]$$
 #The value of y is 'take' (7, 8, 9, 10)

$$y=x[-1]$$
 #The value of y is '!'

$$y=x[-4:]$$
 #The value of y is 'rty!' (-4, -3, -2, -1)

$$y=x[::-1]$$
 #The value of y is ????

```
y=len(x) #The value of y is 32
```

y=x.find("to") # The value of y is 21

indicates a comment.

x[5]='C' #ERROR! (strings are immutable)

Python allows for flexible conversion

A variable of one type can be converted into another type x='2'x=int(x)print(x) x=float(x) print(x) 2.0 x=str(x)print(x) '2.0' x = bool(x)print(x) True x=0x = bool(x)print(x)

False

As long as the data is 'convertible'

Variables and values

Values do not change but the value associated with a variable can change!

```
In [8]: student1 = 87
        id(student1)
Out[8]: 4297151280
                                                                 id: a function that
In [9]: student2 = 95
                                                               returns the location of
        id(student2)
                                                                  a value in memory
Out[9]: 4297151536
In [10]:
        student1 = student2
                                               student1: its value is at the
        id(student1)
                                              same location where student2's
Out[10]: 4297151536
                                                           value is
                                               87: its location in memory is
Out[11]: 4297151280
```

unchanged

Python language elements

Expressions arithmetic, string, logical
Decision making if elif else
Functions
Iteration for, while
Compound data types list, tuple, set, dict

Expressions

An expression is a combination of values and variables, operators, and functions

Python evaluates these expression components following its rules of precedence to arrive at a value

When executed, an expression must evaluate to a value

If an expression doesn't evaluate to a value, Python substitutes a NoneType value None

Let's practice

```
\max(x,y): a function that returns the greater of x and y
```

write a program that inputs 4 numbers into variables x1, y1, x2, y2

store the sum of the larger of each of the pairs x1, y1 and x2, y2 in a variable z

print the value of z

try it!

Data types and expressions

```
x1=5
y1=8
x2="Hello"
y2="Sayonara"
z=max(x1,y1) + max(x2,y2)
print(z)
```

What happens?

Logical expressions

```
Arithmetic expressions: evaluate to int or float String expressions: evaluate to str
```

Logical expressions: evaluate to bool (True or False)

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

Example

```
x1=5
y1=8
x2="Hello"
y2="Sayonara"
z = x1 > y1 and x2 > y2
print(z)
```

This works - even though we're mixing ints and strs because the two operands of the 'and' operator are both boolean

Truth values

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
```

Truth values

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

```
8=x
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 \longrightarrow
       #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
```

Truth values

Logical expressions are evaluated only to the extent necessary to determine their truth value

```
x=8
y=0
result = x and y #0 because y is evaluated last
result = x or y #8 because if x is True then y doesn't matter
result = y and x #0 because y is False and x doesn't matter
```

Logical and relational operators examples

```
x=5
y=8
k=3
p=0
bool(x)
bool(k)
not bool(x)
x==5 and y==8
x==5 and y<8
x==5 or y<8
not (x==5 or y<8)
x==5 and y
x<2 and y and z
x or y
p and x/p</pre>
```

Control flow

program control flow

Logical expressions are used to control program flow

Position rules:

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

program control flow

```
purchase_price = float(input("Purchase price? "))
price_now = float(input("Price now? "))
if price_now < .9 * purchase_price:
    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")

this gets done only if the first logical
    expression evaluates to True

is True
```

if ... elif else

```
if condition1 :
                    if condition 1 is True then the program does statements 1_1 to 1_n
   statement1 1
   statement1 2
                    and jumps to the post if statements
   statement1 n
elif condition2 :
                    if condition 1 is False then condition 2 is checked. If it is True, then
   statement2 1
                    the program does statements 2 1 to 2 n and jumps to the post if
   statement2 2
                    statements
   statement2 n
elif condition3 :
   statement3 1
                    And so on
   statement3 2
   statement3_n
                    If neither of the if or elif conditions evaluate to True then, and only
else:
   statement4 1
                    then, statements 4 1 to 4 n are executed
   statement4 2
   statement4_n
post_if_statement1
post_if_statement2
```

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")
                                                this is a nested block. note the
                                                     additional indenting!
```

Try this

Write a program that:

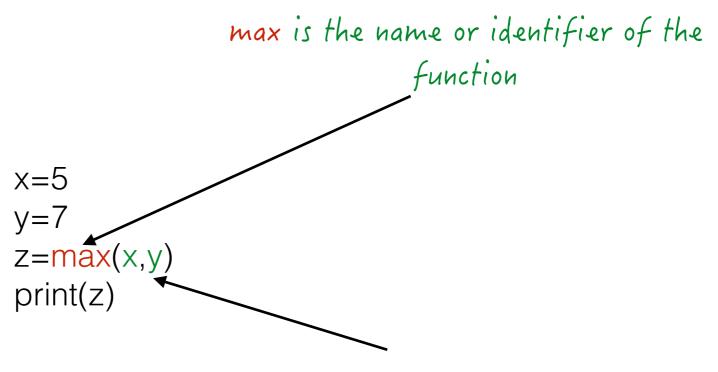
```
1. Inputs the name of a student
2. Inputs the end-of-semester score for that student
3. Prints the grade that the student receives:
    1. if score >= 98: H+
    2. if score >90: H
    3. if score > 80: HP
    4. if score > 70: P
    5. else: F
Example:
```

Example:
Jack

97
Jack: H

Functions

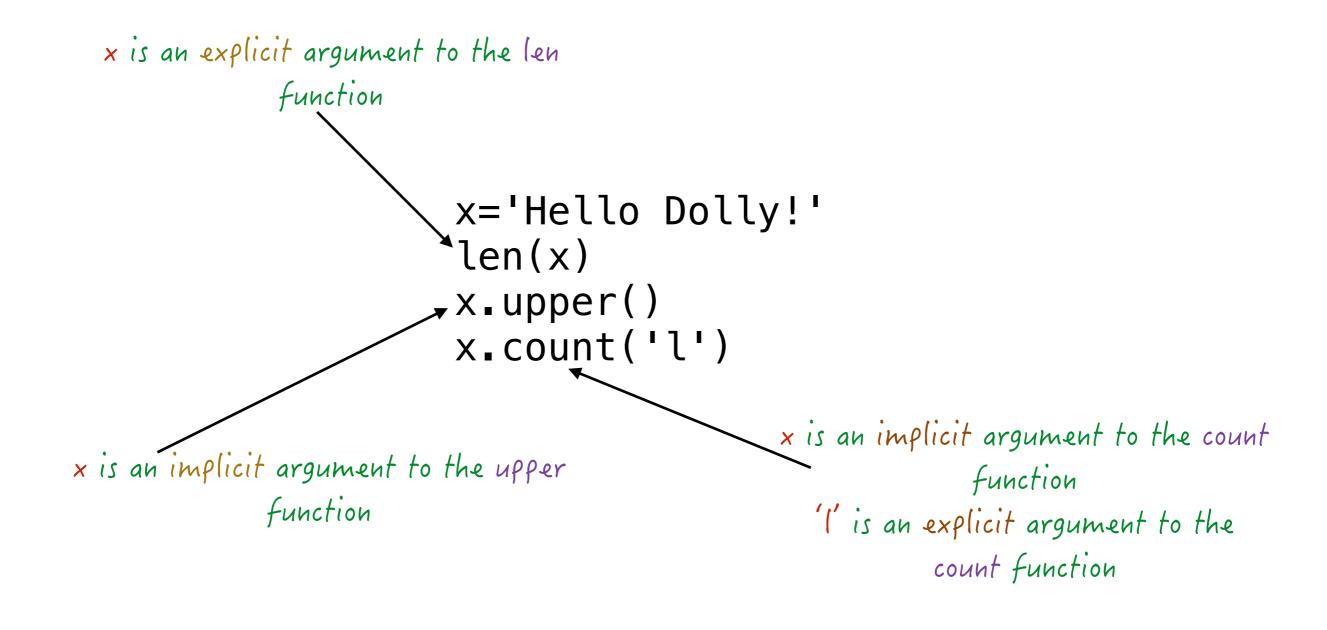
max



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!)

implicit vs explicit arguments



Functions can be grouped in libraries

Libraries need to be imported into a program

math is a library. the

program sets up a

pointer to math

x=74

math.sqrt(x)

sqrt is a function in the

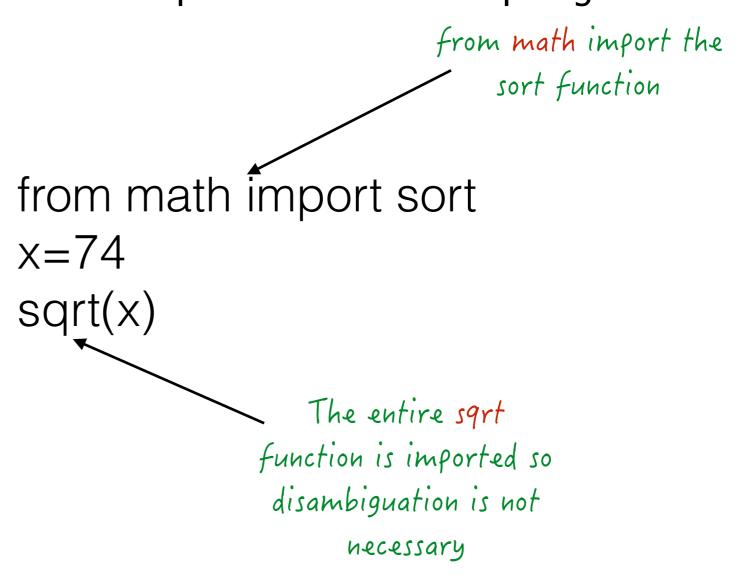
math library and

it needs to be

disambiguated

Functions can be grouped in libraries

Libraries need to be imported into a program



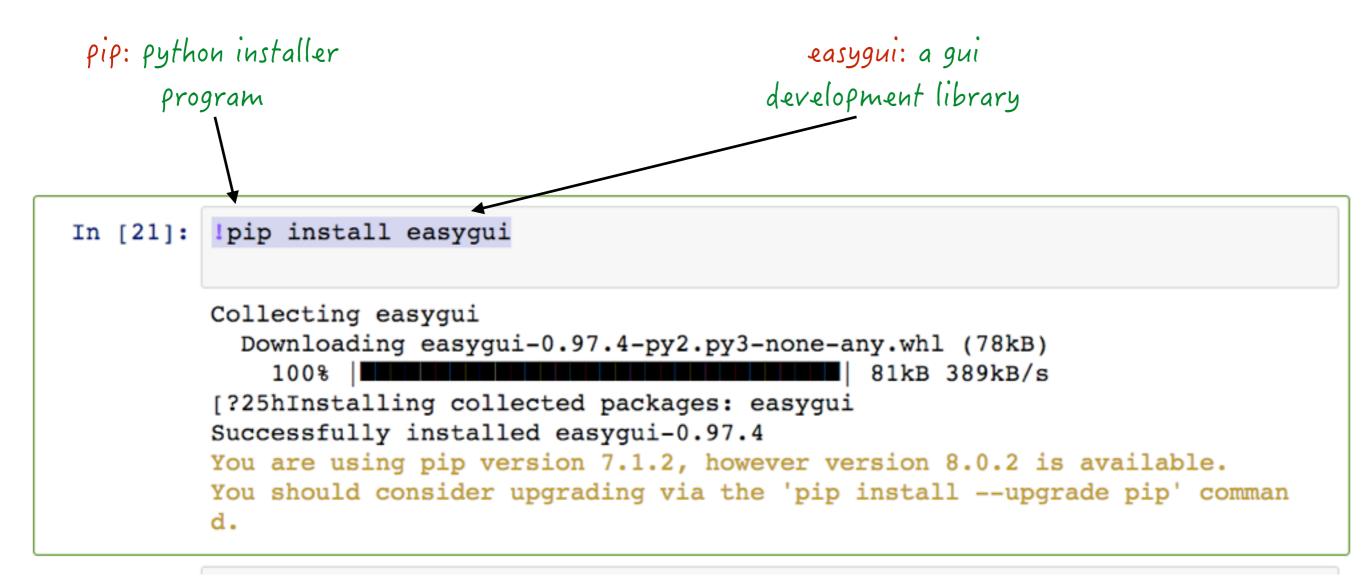
Python is an open source language

With many libraries

Most need to be explicitly installed on your computer

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

Installing Libraries



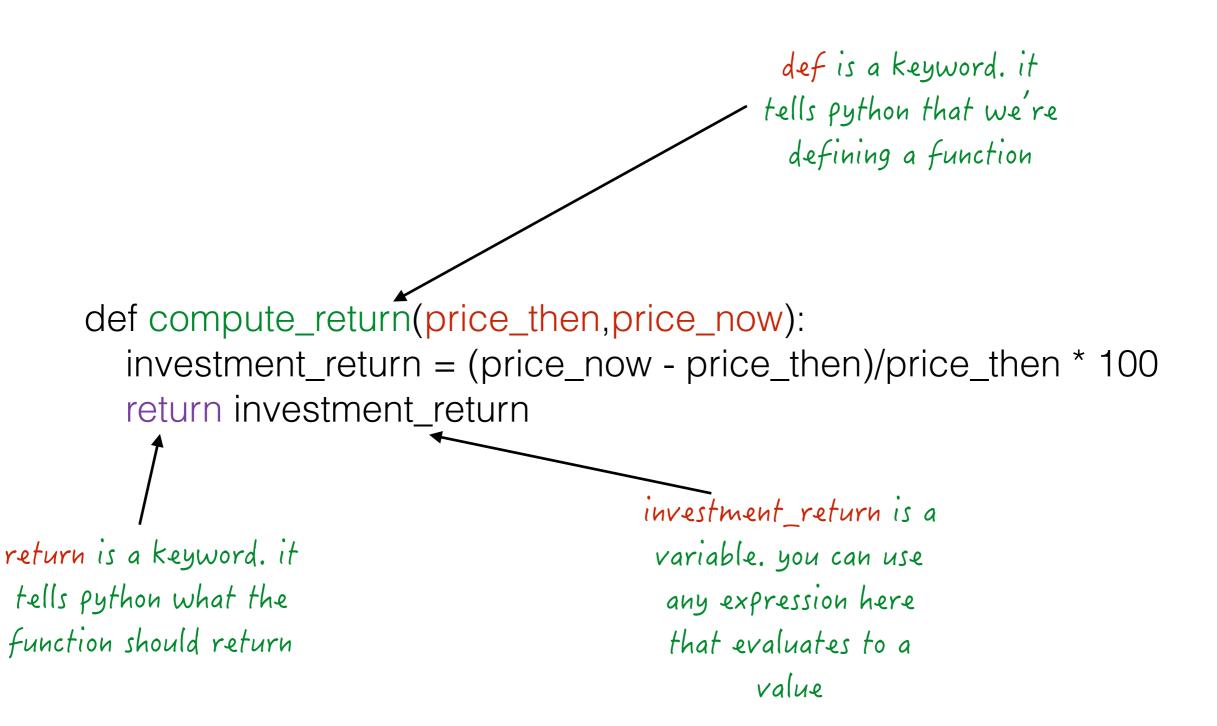
pip is an independent program and can be run through PyCharm or directly from cmd or terminal. Anaconda ipython notebook is the hassle free way of installing libraries

easygui is a library. the program sets up a pointer to easygui but will use the name eg instead

import easygui as eg eg.msgbox('To be or not to be','What Hamlet elocuted')

msgbox is a function in the easygui library and it needs to be disambiguated using whatever name our program gave to the library

Writing functions



return

Every function returns something. If there is no return statement, python uses None

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

print(spam(5)) ---> None
```

arguments

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
```

arguments

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
```

multiple return values

A function can return 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

default arguments

functions can have default arguments

```
O is the default for z
def compute_return(x,y,z=0).
  investment_return=(y-x)/x
  if z and z==100:
    investment_return * 100
  return investment_return
                                                           z is O
                         r1 = compute_return(1.2,91.2)
                         r1 = compute_return(1.2,91.2,100)
```

function arguments

arguments to a function can be other functions

