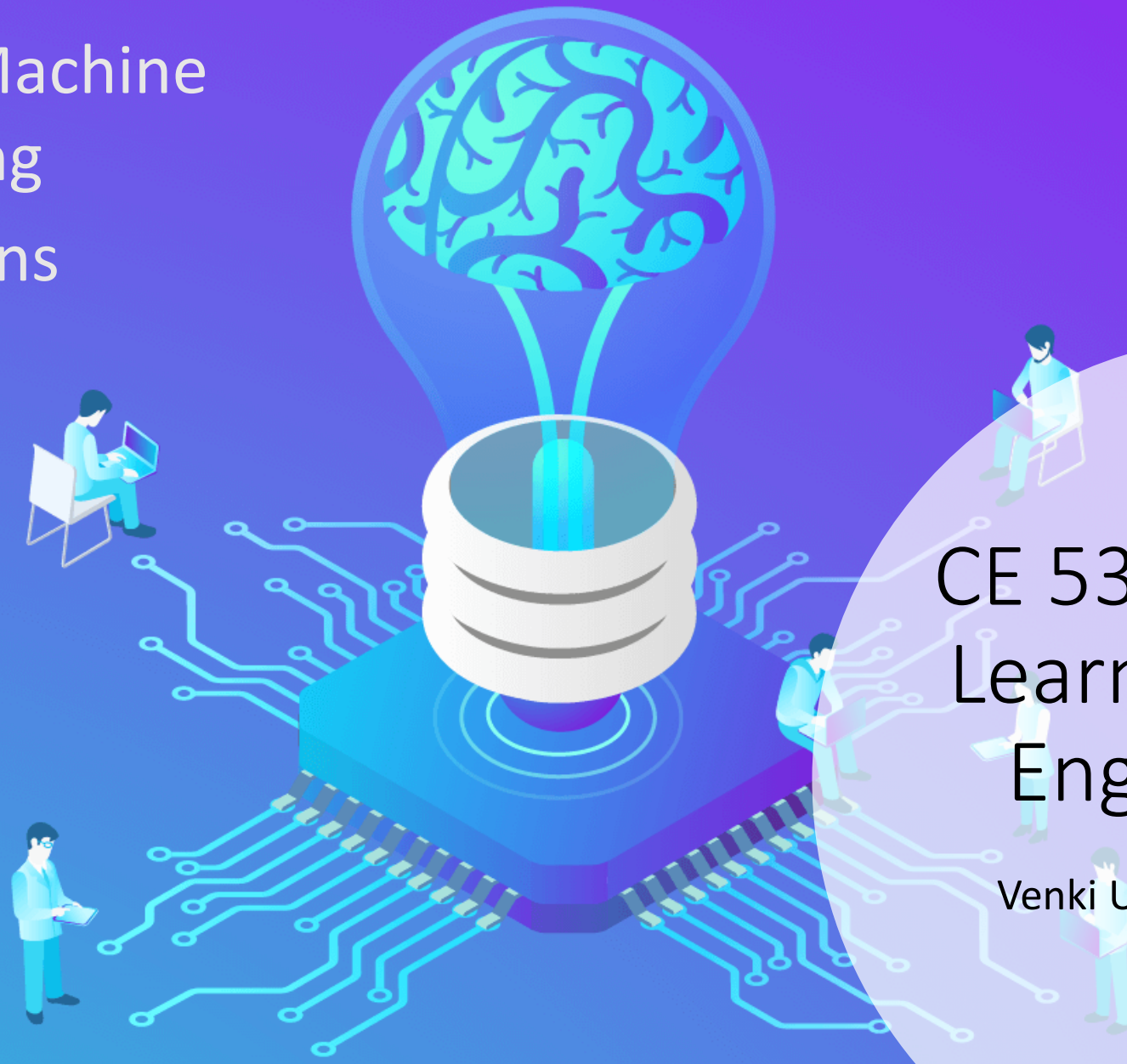
Python for Machine Learning

Functions



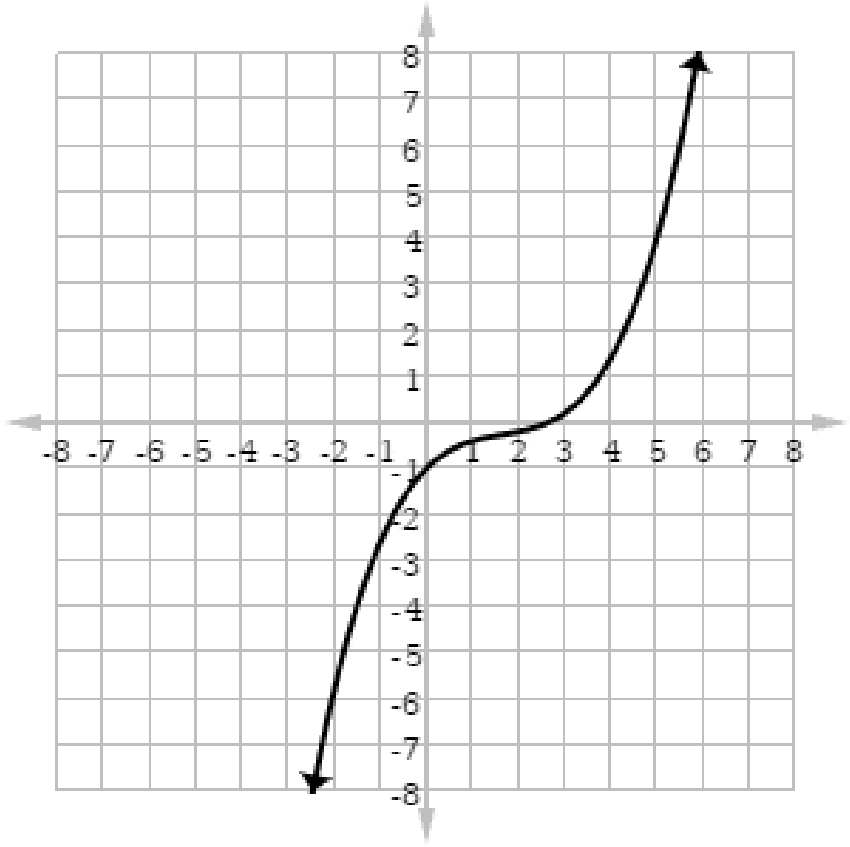
CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap and Goals

- Installed Python and Anaconda Environments
- Introduction to Python
  - Setting working directory
  - Adding comment lines
    - Docstrings
- Introduction to Pandas
  - Reading a csv
  - Extracting columns (attributes)
  - Extracting rows
  - Obtaining summary measures
- Control Statements
  - If, if-elif-else, if-else
  - For loop
  - While loop
  - Use of Boolean operators

Goal of this module is to explore user-defined Functions and methods in Python



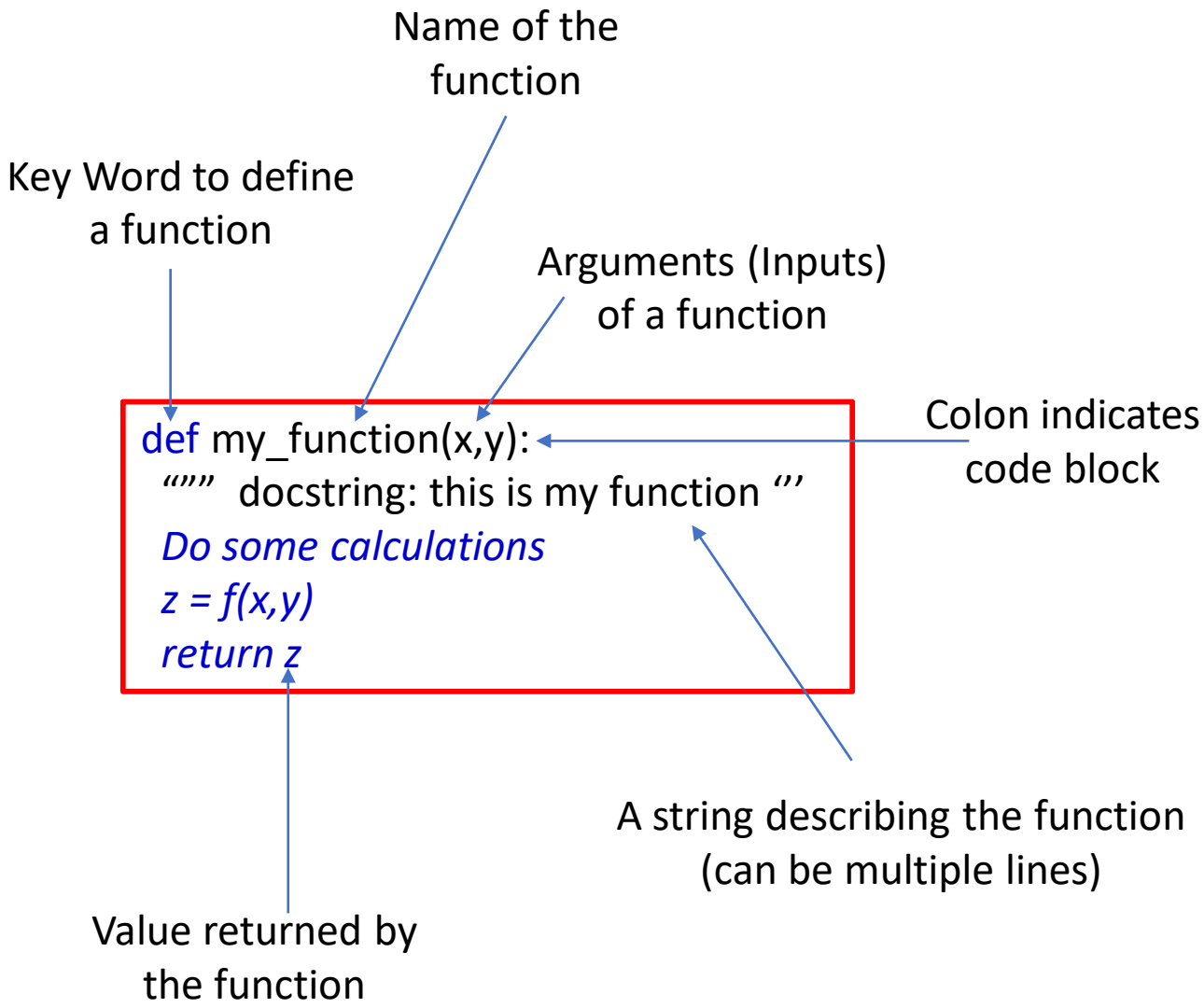
What are Functions

Functions

- Functions are code snippets that perform a specific task
  - All functions **return** a value
  - A function sometimes might return a null value
- Functions are often used to reuse code
  - Write the function once and call it (reuse) many times in your script
- There are essentially three broad classes of functions in python
  - Built-in functions within Python
  - Functions imported using modules
  - User-defined functions

Function Syntax

- Functions have specific syntax that needs to be followed
- Python style guide recommends
  - Functions names be lower case
  - Use _ (underscore) to connect if a function name has two words
- A function cannot be empty
  - At least put a *pass statement in it to avoid error*



Functions

As python is an interpreter a function is available once it is read into the memory

- Therefore useful to put it up in the script
- Make sure libraries (modules) used in the function are placed before it

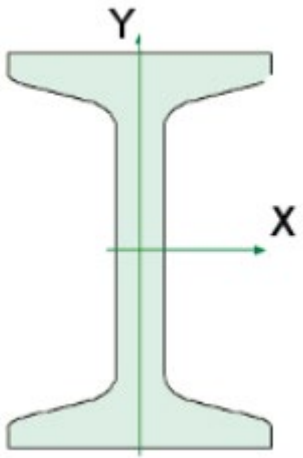
Python also allows recursion

- A function can call itself
- Avoid if possible as it can lead to infinite loops

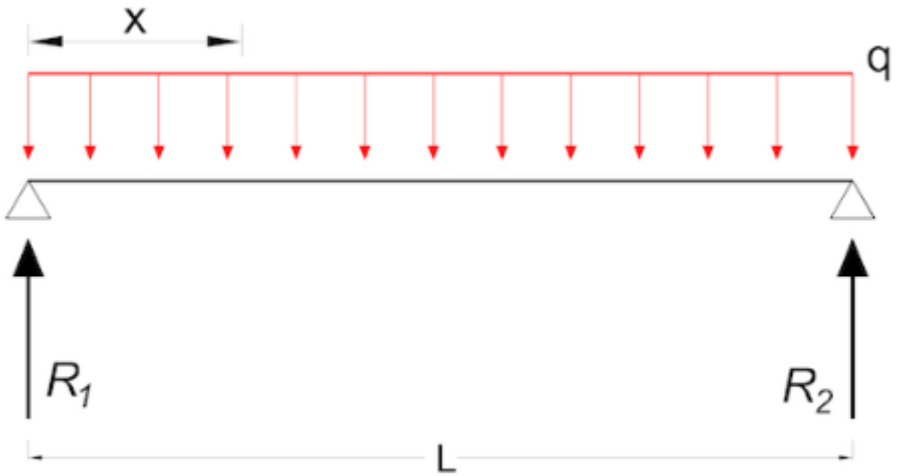
Functions can be passed as an argument to another function

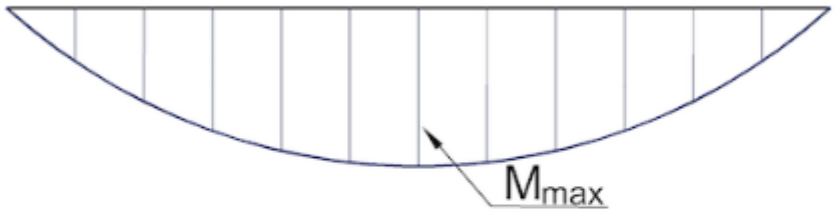
Function Example

- Write a function to calculate the maximum deflection on a steel wide flange beam.
  - Use the function to calculate the maximum deflection on a "W 12 x 35" Steel Wide Flange beam,
    - *100 inches* long, moment of inertia *285 in^4*, modulus of elasticity *29000000 psi*, with uniform load *100 lb/in*



$$\delta_{max} = \frac{5qL^4}{384EI}$$





Function Example (code):

Function Definition

```
def maxdef(L,E,I,q):
    """ Maximum Deflection of a simply supported beam with
uniform load
        L = Length; E = elastic modulus; I= moment of Inertia; q =
uniform loading
    """
    delx = (5*q*L**4)/(384*E*I)  # do not use del as it is python
keyword
    return round(delx,4) # retrun the value
```

Calling a Function in a Script

```
L = 100
E = 29000000
q = 100
I = 285
defs = maxdef(L,E,I,q)
defs
```

A Function needs to be defined first before it is called into a script

Calling Function – Positional and Keyword Arguments

- Functions can be called in two ways
  - Specify the values corresponding to their position
  - Use keywords to specify arguments
    - Positions need not be the same

```
def my_function(x,y):
  """  docstring: this is my function '''
  Do some calculations
  z = f(x,y)
  return z
```

```
A = my_function(3,2)
```

Positional Call: x = 3 and y = 2

```
A = my_function(y=2,x=3)
```

Keyword call x=3, y = 2 but position doesn't matter

Function – Returns Multiple Values

- Unlike many other programming languages, python allows you to return multiple values easily
  - In most languages they have to be combined into a single object before being returned
    - R uses cbind or list to combine and send the results

- Modify the function **maxdef** to include maximum stress on the beam which can be calculated using the following equation

$$\sigma_{max} = \frac{y_{max} q L^2}{8I}$$

Where y_{max} is the distance from the extreme point off the neutral axis (i.e., flange depth/2)

For a W12 x 35 Wide Flange Beam
Depth = 12.5” so y_{max} is 6.25”

http://products.grantsteelcompany.com/viewitems/steel-beams/ries-steel-beams-steel-wide-flange-beams-1

Code – Multiple Values

Function Code:

```
def maxdefstr(L,E,I,q,ymax):
    """ Maximum Deflection and stress of a simply supported
beam with uniform load
        L = Length; E = elastic modulus; I= moment of Inertia; q =
uniform loading
        ymax = flange depth/2"""
    delx = (5*q*L**4)/(384*E*I)  # do not use del as it is python
keyword
    sigm = (ymax *q *L**2)/(8*I)
    return round(delx,4), round(sigm,3) # retrun the value
```

Function returns two values – delx and sigm

A tuple is an immutable list (data structure) in Python

Calling it in a Script

```
L = 100
E = 29000000
q = 100
I = 285
ymax = 6.25
resultx = maxdefstr(L,E,I,q,ymax)
defx, strx = maxdefstr(L,E,I,q,ymax)
```

Resultx: (0.0158, 2741.228) ← **A Tuple**

defx
Out[135]: 0.0158

strx
Out[136]: 2741.228

Unpacked Tuple

Passing Variable Number of Arguments

- In some instances the number of arguments to be passed to a function can vary each time the function is called
  - Calculate the mean of a list
    - The list can be of variable length each time
- Python provides a mechanism to pass variable number of arguments using the ***arg** construct

Two Engineers – **Lazy L** and **Active A** take measurements of a hot-mix asphalt (degree F) as shown below:

A = [280.3, 278.5,284.6, 278.5, 287.4, 268.7]
L = [277.5, 280.4, 286.5]

Write a function to compute the average temperatures obtained by these engineers

Code

Use the ***args notation to pass variable number of arguments** to the function

```
def tempfs(*tempx):
    ''' show variable length input example '''
    xb = sum(tf)/len(tf) # Use built in sum and len func
    xb = round(xb,2)
    return(xb)
```

If you have both fixed and variable length arguments to a function, then place the variable list in the end (after fixed length variables)

Data can be a tuple or a list

```
A = (280.3, 278.5,284.6, 278.5, 287.4, 268.7) # tuple
L = [277.5, 280.4, 286.5] # list
```

Use the ***args notation with the function call** to unpack the tuple or the list

```
tempfs(*A) # A tuple being unpacked before sent to the function
tempfs(*L) # A list being unpacked before sent to the function
```

Passing a Function to a Function

- As functions are objects they can be passed as arguments to other functions
- Simply call the function with its name
  - Arguments for the called function have to be passed separately

```
# Function 1 (sum two #)
def funct1(x,y):
    z = x + y
    return(z)
```

```
# Function 2 (multiply two #)
def funct2(x,y):
    z = x*y
    return(z)
```

```
# Function 3 (call func1 or func2)
def funct3(m,n,o,func):
    zz = func(n,o)
    za = zz + m
    return(za)
```

You can always call available functions within a function without passing them explicitly

```
funct3(2,3,4,funct1) #call function 1
funct3(2,3,4,funct2) #call function 2
```

Anonymous – Lambda Functions

- Anonymous functions
  - Use lambda keyword
  - Are a single line function
  - Usually but not necessarily a single variable function
  - They can be passed as functions to other functions

A lambda function with two variables

```
zza = lambda x,y: x*y
zza(2,3)
```

```
def sq (x):  # Function to square a value
   z = x**2
   return(z)
```

=

```
sql = lambda x: x**2    # Equivalent Anonymous Function
```

```
sq(2)  # returns 4

sql(2) # Also returns 4
```

Anonymous functions are useful to transform data

Currying Functions

- Currying – Deriving a new function using an existing function through partial argument application
  - Named after Haskell Curry
  - Pass some arguments of a generic function to get a specific (Curried) function

Generic Function area (Area of a Rectangle x and y)

```
# Function to compute area of a rectangle
def area(x,y):
    a = x*y
    return(a)
```

Curried Function asq (Area of a Square x = y)

```
asq = lambda x: area(x,x) # Area of a square
```

Variable Scope

- A variable can have either local or global scope
- A variable defined within a function has local scope
  - It is only available when the function is executing
  - It is removed from the memory once the function has finished executing

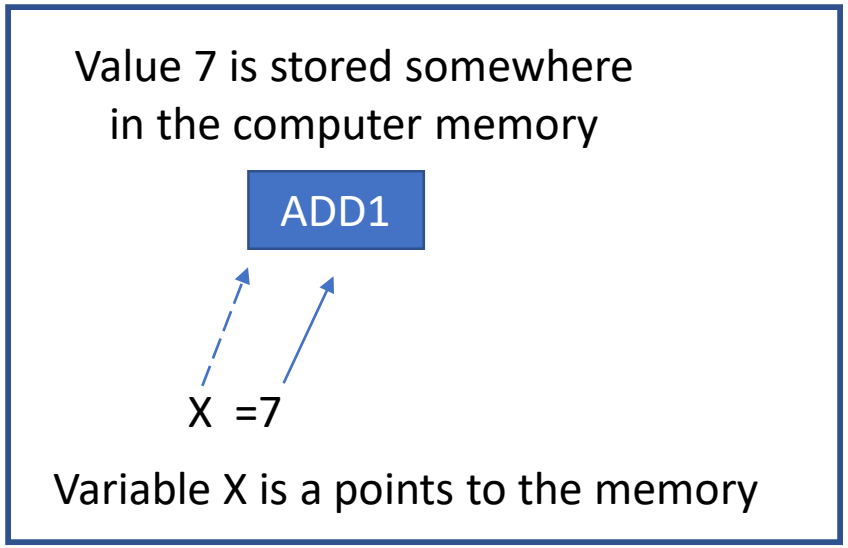
```
def locscp(x):
    y = 2  # local in scope
    z = x + 2
    return(z)

a = 10
locscp(a)
y
```

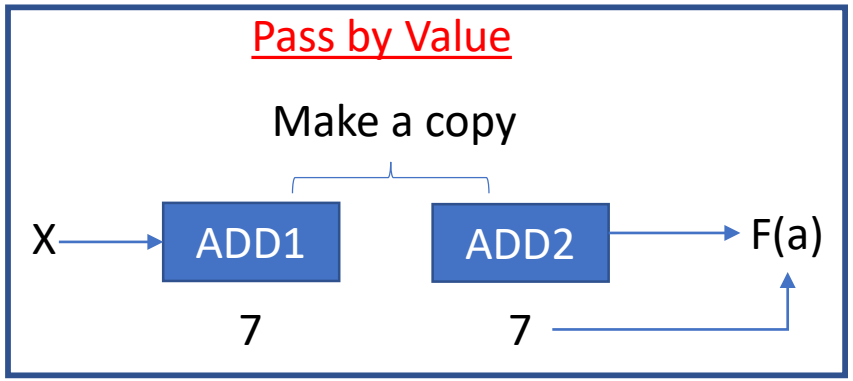
Console

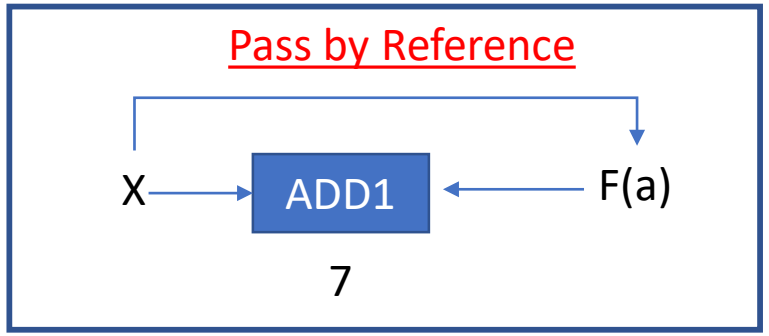
```
def locscp(x):
    y = 2  # local in scope
    z = x + 2
    return(z)


a = 10
locscp(a)
y
Traceback (most recent call last):

  File "<ipython-input-40-3f8ba2241f14>", line 9, in <module>
    y

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

Passing Values to Functions

- Generally values to functions are passed in two ways
  - Pass by value
    - The value is copied to a separate location and passed to the function
    - Safe as the original value cannot be destroyed
    - Memory intensive as a copy is made
  - Pass by reference
    - The reference (memory) location of the value is passed
    - There is a chance that the original value is tampered by the function
    - Memory efficient

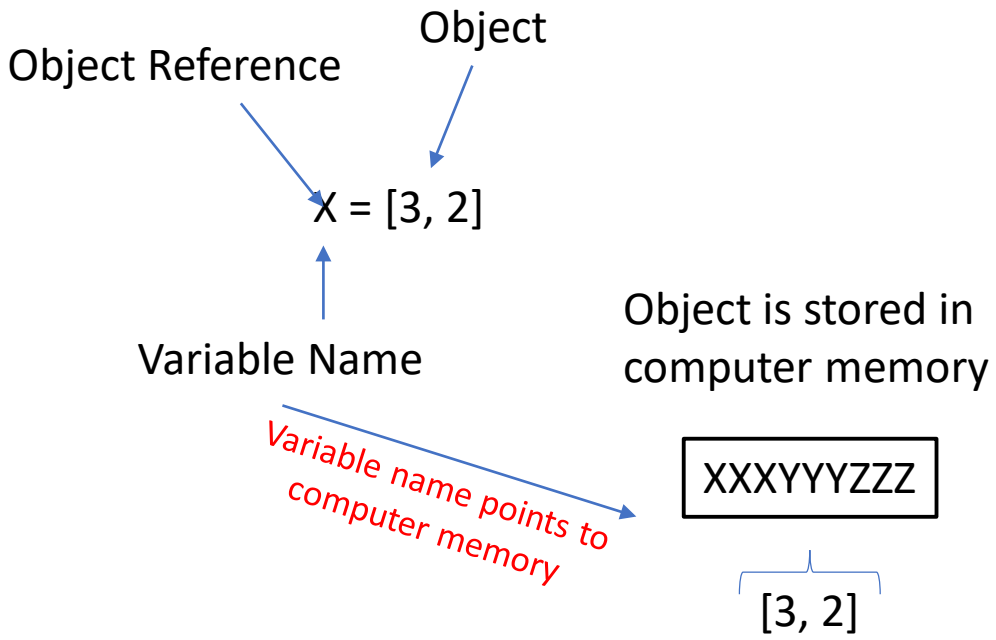






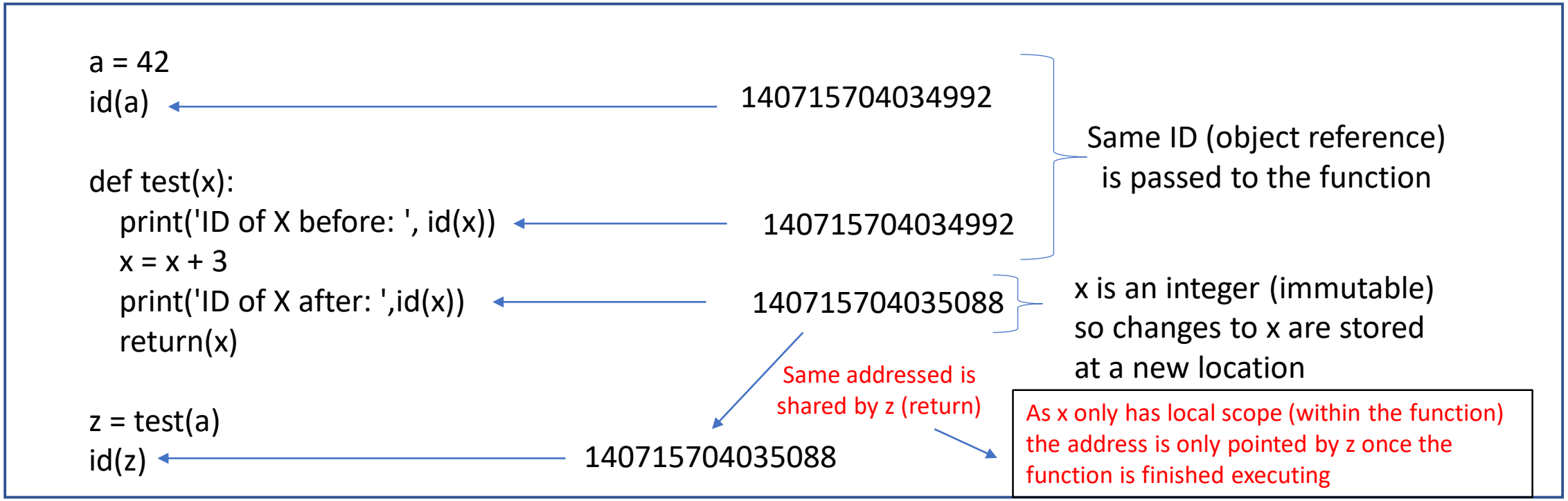
Python – Pass by Object Reference

- Python neither passes by value or passes by reference
  - In some ways it does both
- Python passes by object reference
  - A variable name is a reference to the object
- Python passes the object reference
  - What happens to the reference depends upon whether the object is **mutable** or **immutable**

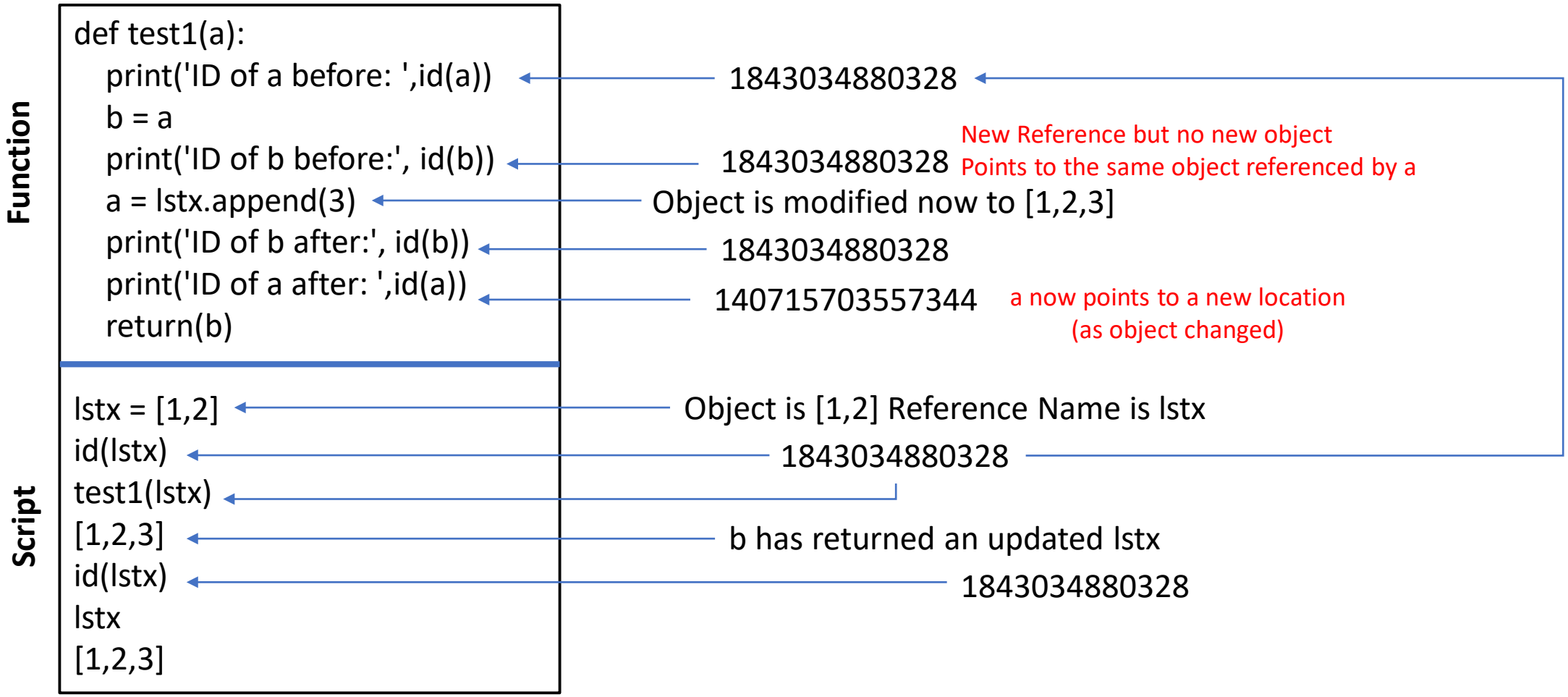


Python – Pass by Object Reference

- Python has a built-in function **id** which can be used to uniquely identify an object
  - Now two objects can share the same memory, so id is analogous to memory



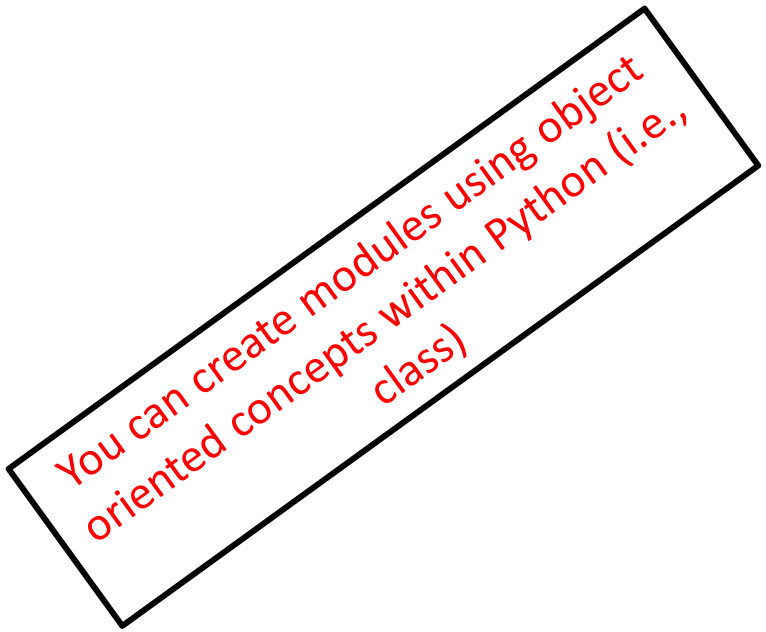
Python – Pass by Object Reference



As a list is mutable it can be changed within a function – For example, **lstx is changed by the function above**
Multiple Names (variables) can point to the same object (a, lstx, b) all point to the same object (list)
When an object is changed all names now point to the updated object – object changed from [1,2] to [1,2,3]

Modules

- Modules are functions that are tagged to objects
  - These functions can be accessed using the **obj.mth** type syntax
- As python methods are associated with an object it has access to the data associated with that object
- Methods can modify the object
  - Functions usually don't



You Should Know

- What are functions
- How pass a fixed set of arguments to functions
- How to write and pass a unknown number of arguments to functions
- How to pass functions to functions
- Lambda or anonymous functions – what are they? And how to write them
- How functions affect mutable and immutable data
  - Pass by object reference
- What are methods