NYU TANDON SCHOOL OF ENGINEERING

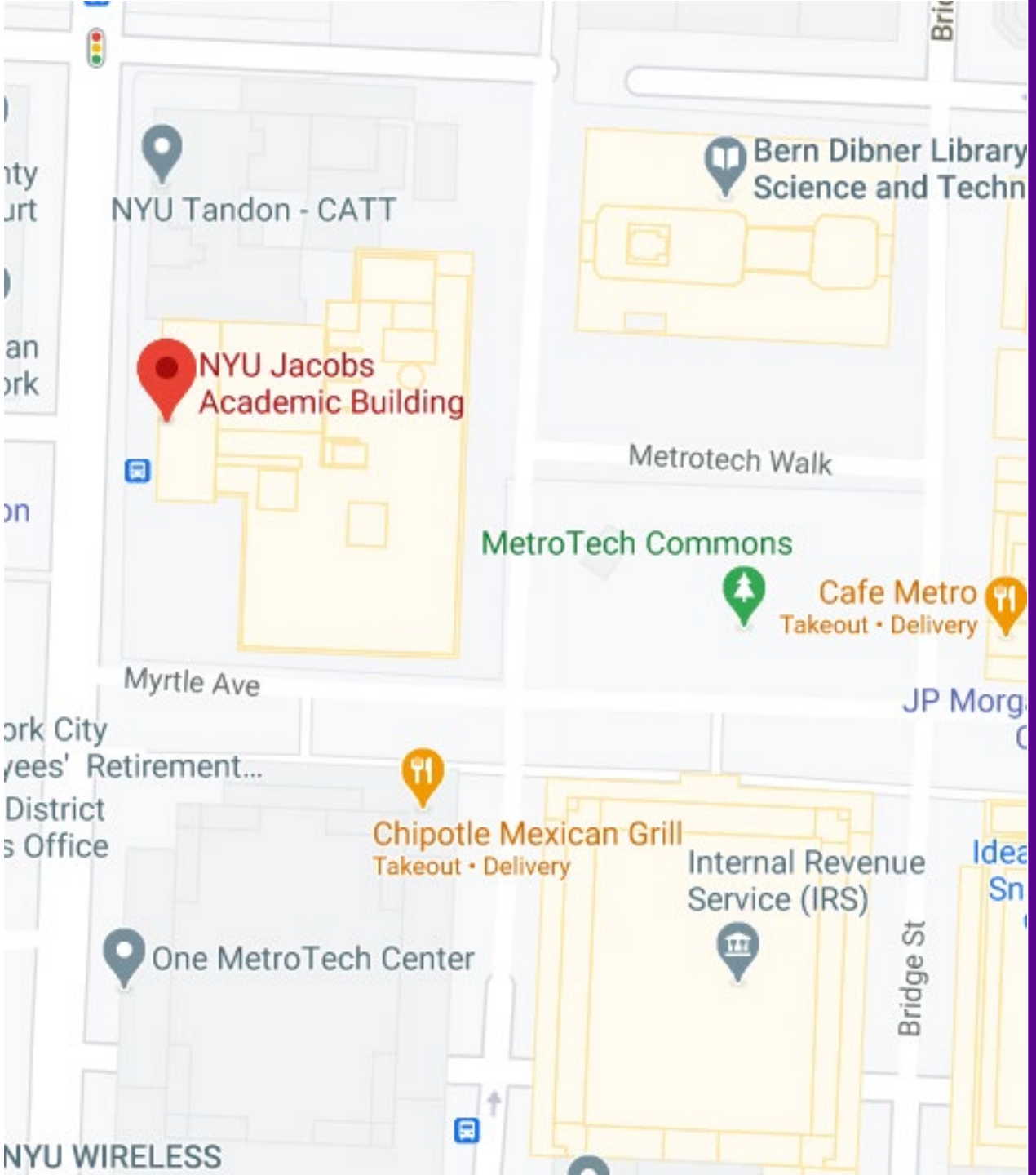
Lecture 2
Packaging Code

Programming for Business Analytics
MG-GY 8401

NYU TANDON SCHOOL OF ENGINEERING

Agenda

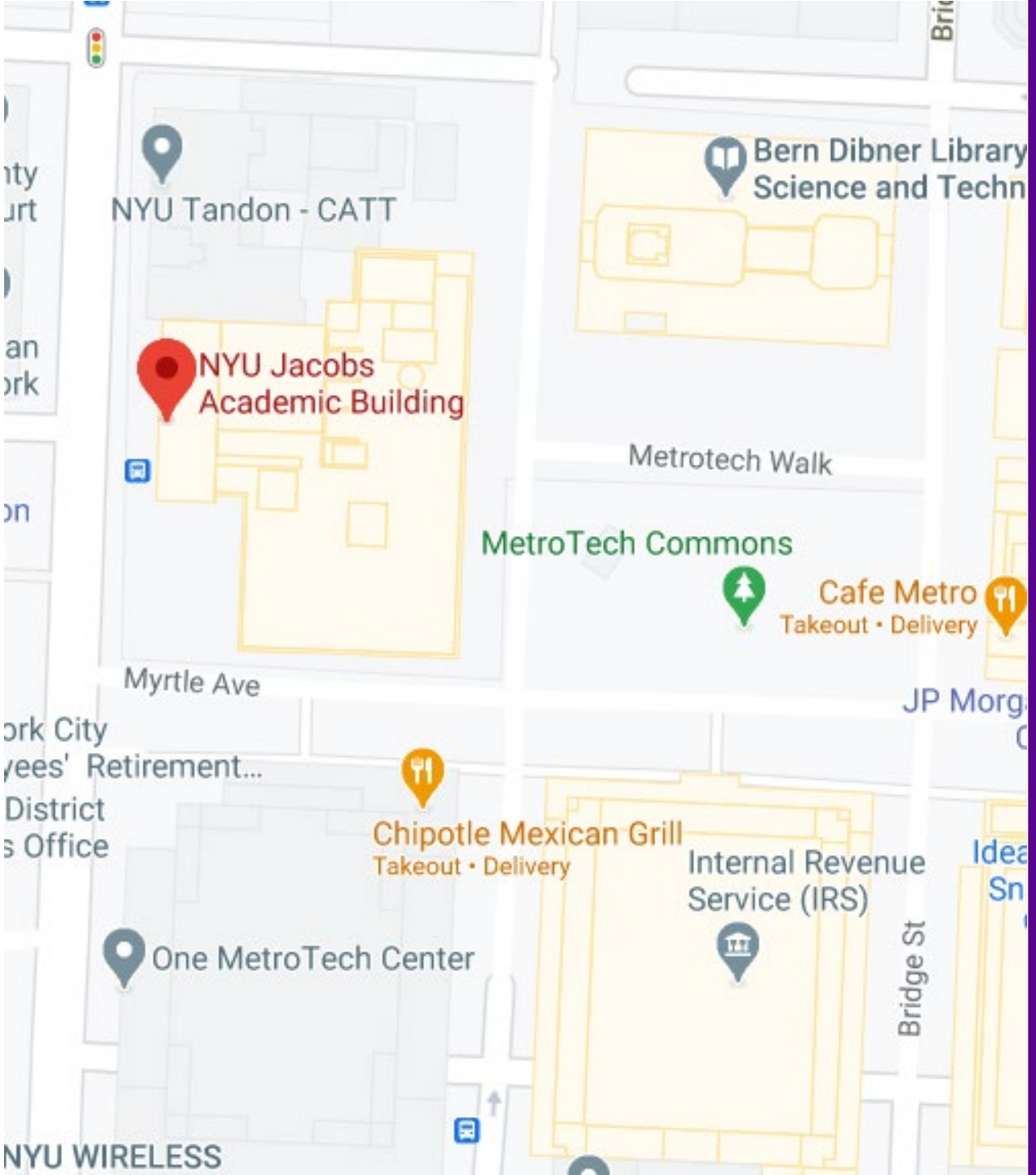
- Function
- Classes
- Modules



NYU TANDON SCHOOL OF ENGINEERING

Logistics

- NYU LMS
  - Python Programming
- JupyterHub
- Slack
- Gradescope



NYU TANDON SCHOOL OF ENGINEERING

Logistics

- Homework
  - Homework 0
  - Homework 1
  - Homework 2

Factory Workers Become Coders as Companies Automate

Employees who show aptitude are gaining new skills, helping businesses make better-quality products and bringing in more revenue

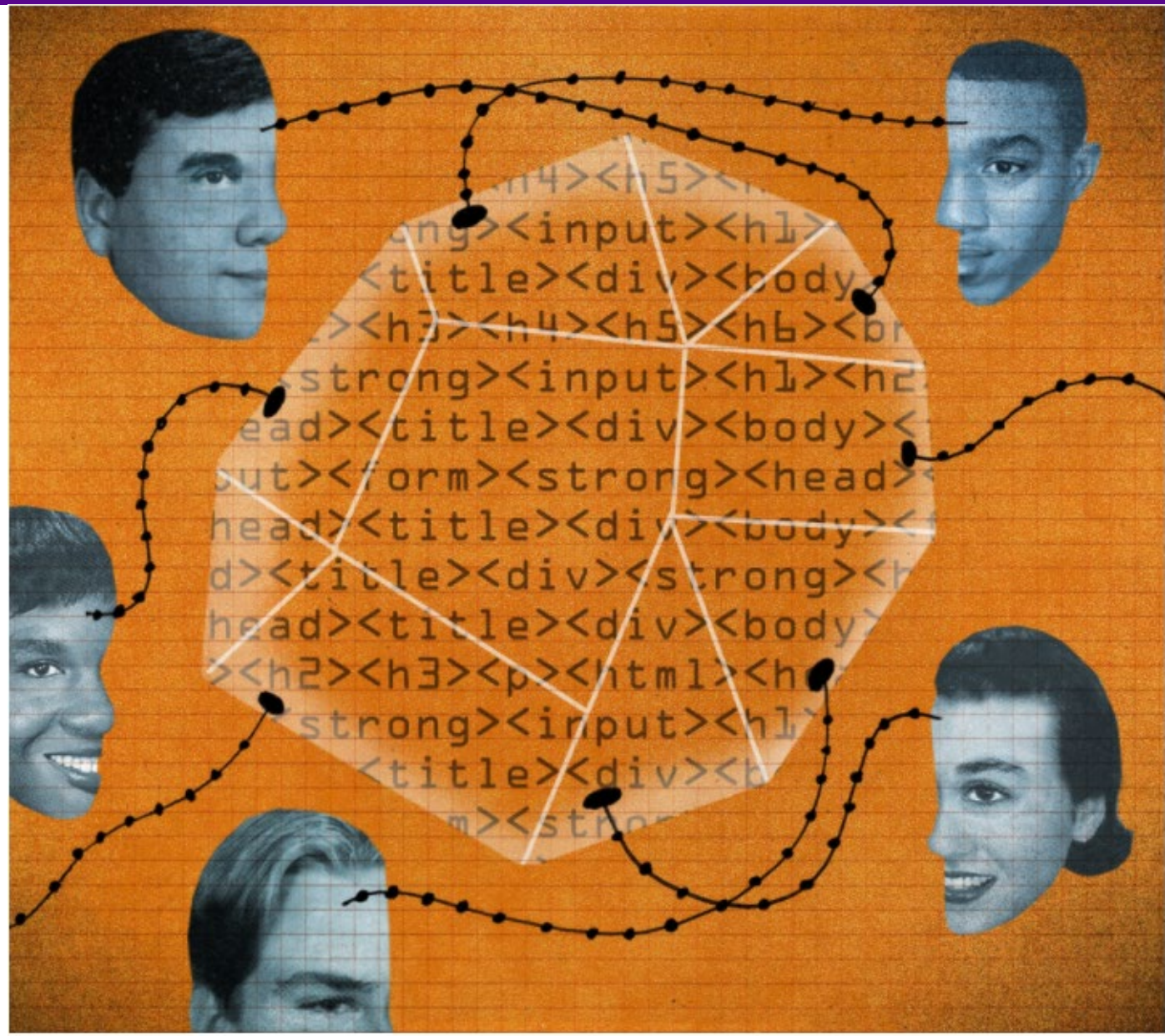
By Agam Shah
May 17, 2019 5:30 am ET

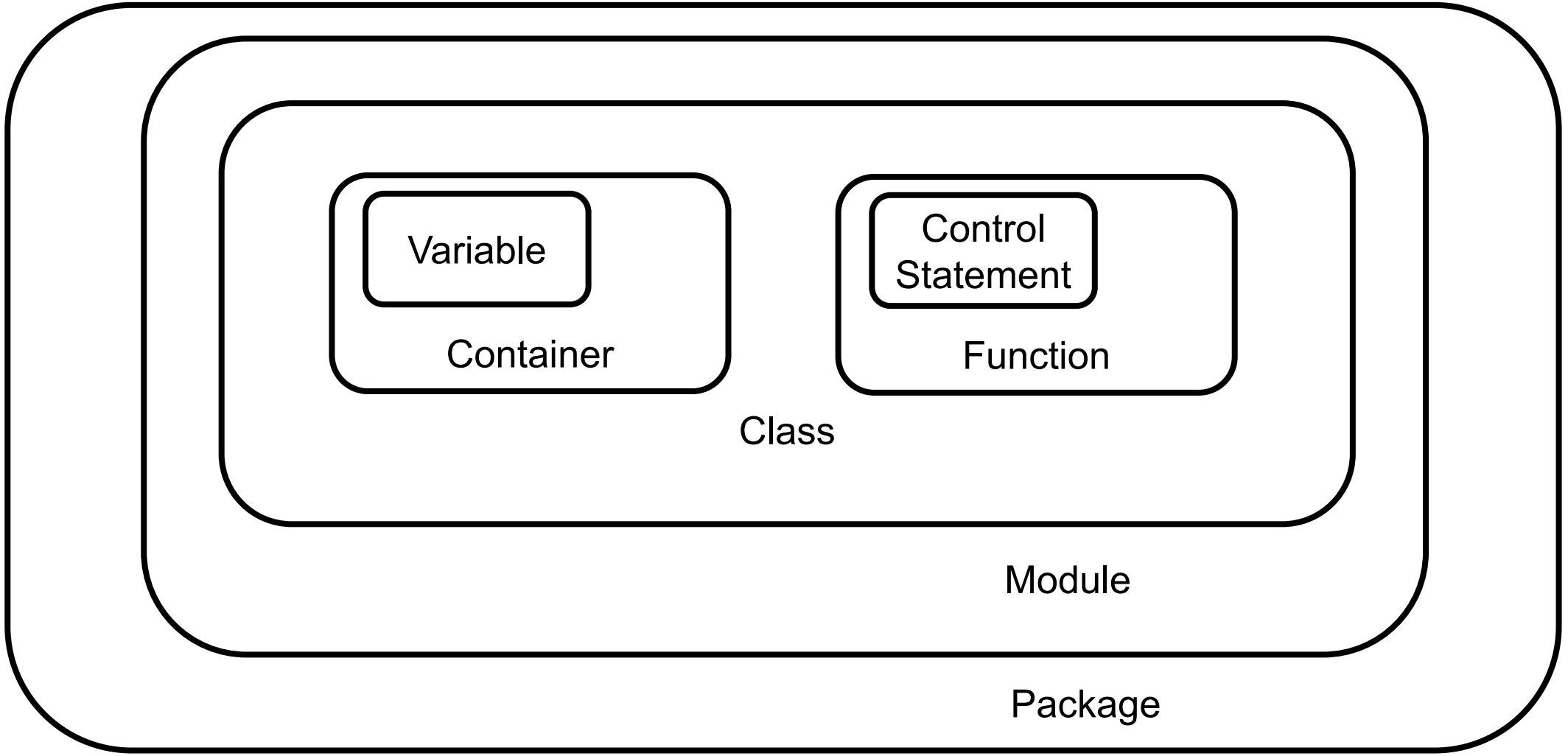
PRINT TEXT

As automation changes the way factories operate, some U.S. companies are training workers in programming and robotics, letting machinists get a taste of coding.

The company's machinists develop code so robots can make parts to specifications, replacing several workers who physically created parts. Other employees use collaborative software to interact with customers on real-time design changes, helping the company manufacture higher-quality steel products, charge more for them and create unique intellectual property, he said.

"We're not going to beat the competition because we are charging lower prices. We are going to beat the competition because of the technology. These are factory workers turning into coders to exploit the technologies"




Variable
Container
Control Statement
Function
Class
Module
Package

What is the output of the following code?

A. [1,0,4,5,2,3,3,1,6,5]
B. [1, 0, 4, 5, 2, 3, 6]
C. [0, 1, 2, 3, 4, 5, 6]

```
input_list = [1,0,4,5,2,3,3,1,6,5]

output_list = []
for entry in input_list:
    if not entry in output_list:
        output_list.append(entry)

print(output_list)
```

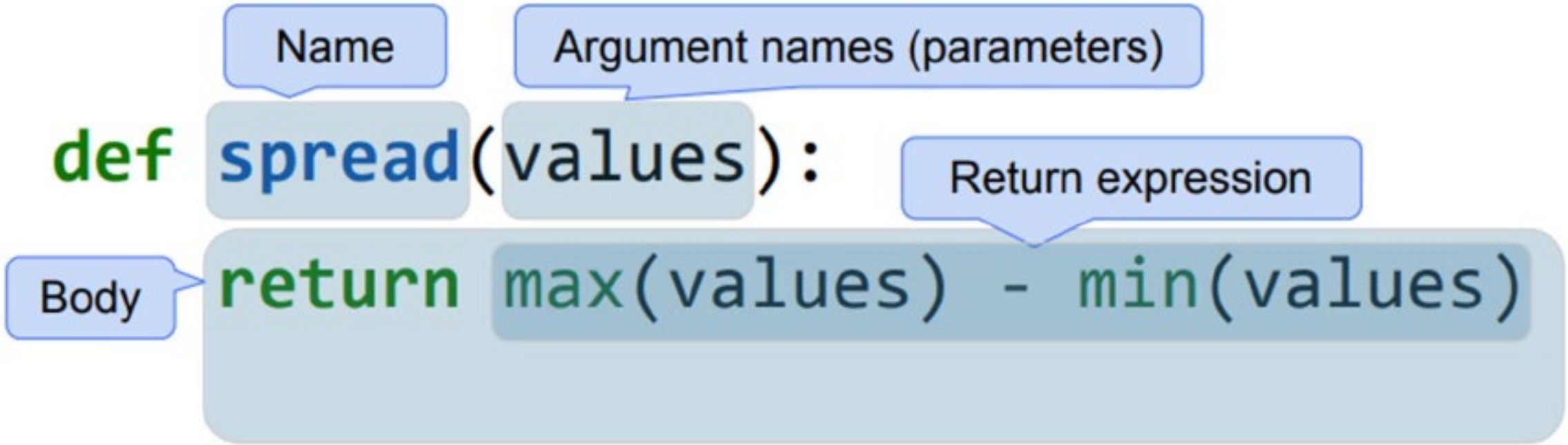
What is the output of the following code?

A. [1,0,4,5,2,3,3,1,6,5]
B. [1, 0, 4, 5, 2, 3, 6]
C. [0, 1, 2, 3, 4, 5, 6]

```
input_list = [1,0,4,5,2,3,3,1,6,5]

output_list = []
for entry in input_list:
    if not entry in output_list:
        output_list.append(entry)

print(output_list)
```


Name
Argument names (parameters)
def spread(values):
Return expression
Body
return max(values) - min(values)

- The `def` keyword indicates a block of code for a function

```
def my_function(arg1, arg2):
  x = arg1 + arg2
   return x * 3
```

- The `return` keyword indicates the output of the function

- We can add comments to functions through doc-strings
- We use strings spanning multiple lines to provide information about the function particularly the function signature

```
def multiply(a,b=1):
    '''this is a
       doc-string for the fuction'''
    m = b*a
    return m
```

- The inputs of the function are called parameters

```
def my_func(x, y):
    return x + y
a = 10
z = my_func(4, a)
```

```
a = 10
x = 4
y = a
z = x + y
```

- The values assigned to the parameters are called arguments

```
class ClassExample(object):
    def __init__(self, data_example):
        self.attribute_example = data_example


    def getter(self, n):
        return self.attribute_example[:n]
```

- We can pass arguments to parameters indicating the name of the parameters.
- Keyword arguments help prevent confusion about the order of inputs

```
def multiply(a,b):
    m = b*a
    return m

print(multiply(b = 2, a = 3))
```

Default Arguments

- If a function has default arguments, then the function can take fewer inputs.

```
def func(a, b, c=10, d=100):
    print(a, b, c, d)

func(1,2,3,4)
func(1,2)
```

- Arguments are passed by reference to parameters

```
def changer(x, y):
    x = 2              # changes the local value of x only
    y[0] = 'hi'        # changes mutable object
```

- Assigning other data to the parameter does not impact the argument. However, modifying a the parameter can impact the argument

- Names for variables outside the function never conflict with names inside the `function`

```
x = ‘Fred'
def func():
   x = 'Jane'        # A different name

func()
print(x)
```

Variable Length Arguments

- We can pass varying numbers of arguments to a function using *args

```
def multiply(*args):
    m = 1
    for entry in args:
        m = m * entry
    return m

print(multiply(3))
print(multiply(3,2))
print(multiply(3,2,4))
```

What is the output of the following code?

A.
```
a equals 7
b equals 3
```

B.
```
a equals 3
b equals 7
```

```python
a = 3
b = 7
def swap(a, b):
    temp = a
    a = b
    b = temp

swap(a, b)
print(f"a equals {a}")
print(f"b equals {b}")
```

What is the output of the following code?

A.
```
a equals 7
b equals 3
```

B.
```
a equals 3
b equals 7
```

```python
a = 3
b = 7
def swap(a, b):
    temp = a
    a = b
    b = temp

swap(a, b)
print(f"a equals {a}")
print(f"b equals {b}")
```

- Reuse of Code
- Portability of Programs
- Separate Namespaces

- The `import` statement loads everything from the module into the current module
- The module is run once, regardless of the number of times it is imported
- `import` creates a *module object*, so attributes must be accessed using qualifier notation

```
import module
module.function()
module.attribute = 3
```

- The `from` statement allows individual attributes to be imported into the current module
- It does not import the whole module, only the specific attribute you specify

```
from module import function
from module import a, b, c
function()
print(a)

from module import *  # import everything
```

- Functions and variables in a module are accessible through qualification notation
  - `module.variable`
  - `module.function`
- Modules can import other modules

- Attributes of c can be accessed from module a using the same attribute notation b.c.attribute

- We can bundle modules together with packages.
  - Modules are files
  - Packages are folders
- We need the __init__.py file in the folder to specify the package.
  - While the file can be empty, Python will not recognize the folder as a package without it.

NYU TANDON SCHOOL OF ENGINEERING

Functions

- Package control statements

Classes

- Package containers + functions

Modules

- Package classes

NYU TANDON SCHOOL OF ENGINEERING

References

- Lubanovic, *Introducing Python*

(Chapters 9,10,11)

Questions

- Describe the learning objectives.
- Summarize the relevant take-aways.
- Ask about unclear information.