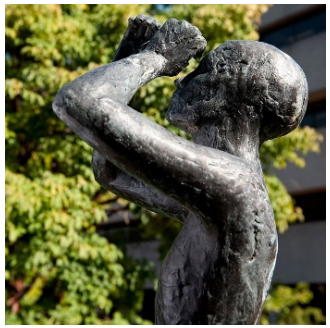


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

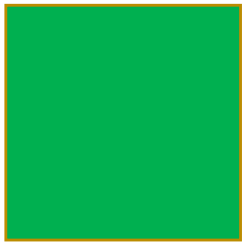
A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland

Coláiste na hOllscoile Corcaigh

ZIP



Zip – takes two equal length iterable collections

(list, string, tuple)

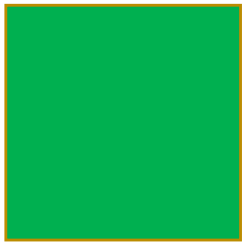
And merges them into a list of pairs of tuples:

List Example:

```
a = [1, 2, 3, 4, 5]
b = [2, 2, 9, 0, 9]
print(list(zip(a, b)))
# [(1, 2), (2, 2), (3, 9), (4, 0), (5, 9)]

print(zip(a, b))
# Python3 - <zip object at 0x105b49f50>
# Python 2 - [(1, 2), (2, 2), (3, 9), (4, 0), (5, 9)]
```

ZIP

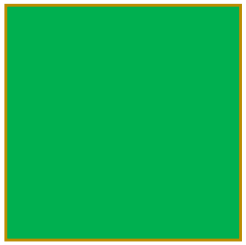


String Example:

```
a = "hold"
b = "were"
print(list(zip(a, b)))
# [('h', 'w'), ('o', 'e'), ('l', 'r'), ('d', 'e')]

print(zip(a, b))
# Python3 - <zip object at 0x10a6f4e60>
# Python 2 - [('h', 'w'), ('o', 'e'), ('l', 'r'), ('d', 'e')]
```

ZIP

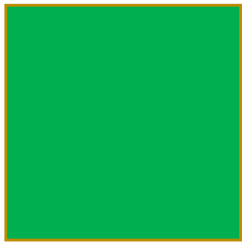


Integer Example:

```
# integer example
a = 123
b = 456
print(list(zip(a, b)))
# zip argument #1 must support iteration

print(zip(a, b))
# Python3 - zip argument #1 must support iteration
# Python 2 - zip argument #1 must support iteration
```

ZIP

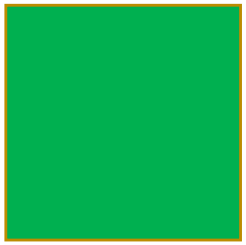


Integer Example Fix:

```
# integer fix:
a = 123
b = 456
print(list(zip([a], [b])))
# [(123, 456)]

print(zip([a], [b]))
# Python3 - <zip object at 0x1094c8e10>
# Python 2 - [(123, 456)]
```

ZIP

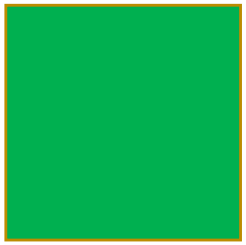


Integer Example Fix:

```
# integer fix:
a = 123
b = 456
print(list(zip([a], [b])))
# [(123, 456)]

print(zip([a], [b]))
# Python3 - <zip object at 0x1094c8e10>
# Python 2 - [(123, 456)]
```

ZIP

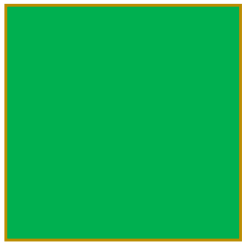


Break String Example:

```
a = "hold"
b = "were"
print(list(zip([a], [b])))
# [('hold', 'were')]

print(zip([a], [b]))
# Python3 - <zip object at 0x10a6f4e60>
# Python 2 - [('hold', 'were')]
```

MAP



Extracts each element in an iterable and passes it to a function

Parameters:

Function and an iterable collection

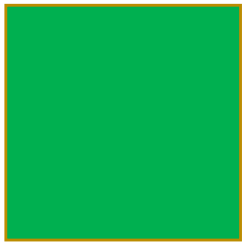
(really it can be viewed as a function and parameters)

Returns a list

```
# # # MAP # # #
a = [1, 2, 3, 4, 5]
b = [2, 2, 9, 0, 9]

print(map(max, a, b))
# [2, 2, 9, 4, 9]
```

MAP



Take each of the values from ‘a’ and ‘b’

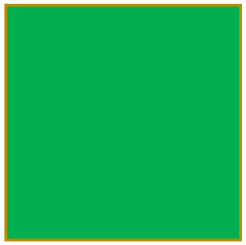
Pass them to the ‘max’ function

Store the returned ‘maximum value’ in a list

```
# # # MAP # # #
a = [1, 2, 3, 4, 5]
b = [2, 2, 9, 0, 9]

print(map(max, a, b))
# [2, 2, 9, 4, 9]
```

MAP



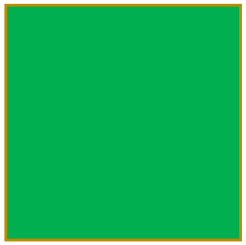
Similar to zip – Python3 assigns map results as an object

```
# # # MAP # # #
a = [1, 2, 3, 4, 5]
b = [2, 2, 9, 0, 9]

print(map(max, a, b))
# Python3 - <map object at 0x10a82c090>
# Python 2 - [2, 2, 9, 4, 9]

print(list(map(max, a, b)))
# Python3 - [2, 2, 9, 4, 9]
# Python 2 - [2, 2, 9, 4, 9]
```

MAP

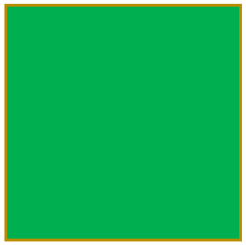


Map returns a result based on the smallest list only

```
a = [1, 2, 3]
b = [2, 2, 9, 0, 9]
print(list(map(max, a, b)))
# [2, 2, 9]

a = [1, 2, 3, 4, 5]
b = [2, 2, 9, 0]
print(list(map(max, a, b)))
# [2, 2, 9, 4]
```

MAP

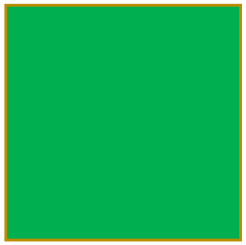


You can create your own functions and single parameters

```
def square(n):
    return(n*n)


a = [1, 2, 3, 4, 5]
print(list(map(square, a)))
# [1, 4, 9, 16, 25]
```

LAMBDA



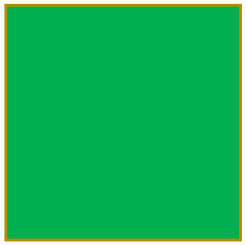
In Map we could call a function and pass each value in a list as individual parameters

```
def square(n):
    return(n*n)


a = [1, 2, 3, 4, 5]
print(list(map(square, a)))
# [1, 4, 9, 16, 25]
```

This function can be called from anywhere in our code

LAMBDA



In Python we can create a function that only exists for a single moment in time

For this we use the keyword ‘lambda’

‘lambda’ is a short way of creating an anonymous function (functions without a name)

Often used for a once-off function such as scenarios where you need to pass the result of a function as a parameter to another function.

LAMBDA

If we were to rewrite the ‘square(n)’ function using ’lambda’

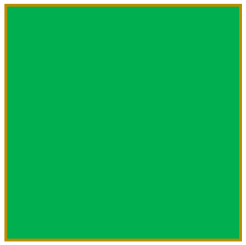
Using the format ‘lambda <parameter> : <expression>’

```
# # # # LAMBDA # # #
def square(n):
    return n*n



a = [1, 2, 3, 4, 5]
print(list(map(lambda n: n*n, a)))
# [1, 4, 9, 16, 25]
```

LAMBDA



Using the format ‘lambda <parameter> : <expression>’

So remove the “def <name>” and “return”

Use ‘map()’ to extract each element from a and pass to lambda

Cast to ‘list()’ for Python3 compatibility

```
# # # # LAMBDA # # #
def square(n):
    return n*n


a = [1, 2, 3, 4, 5]
print(list(map(lambda n: n*n, a)))
# [1, 4, 9, 16, 25]
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh