Columbia Business School
AT THE VERY CENTER OF BUSINESS

Introduction to Python

Session 2

Recall: Data Structures

Dictionary (Dict)

List (List)
Tuple (Tuple)

Boolean (Bool)
String (Str)

Last week

Integer (Int)
Float (Float)

Today!

Finding the Type of a Variable

```
type(variable_name)
```

Deep Dive: Integers & Floats

What is a integer?

Definition

Whole numbers

```
int1 = 6
int2 = -888
int3 = 100000
```

What is a float?

Definition

Floating point number:
number with a decimal point

```
float1 = 6.0000000
float2 = -888.88
float3 = 100000.000000001
```

Converting to Integer

```
print(int(6.0))
```

```
6
```

Python will try to convert the number to an integer

```
print(int(6.75))
```

```
6
```

Python will remove the decimal part (No rounding occurs!)

Converting to Float

```
print(float(6))
```

```
6.0
```

Python will try to convert the number to a float

Converting from Strings

```
print(int('6.75'))
```

?

```
print(int('six'))
```

?

```
print(float('6.75'))
```

?

Converting from Strings

```
print(int('6.75'))
```

```
ValueError
```

```
print(int('six'))
```

```
ValueError
```

```
print(float('6.75'))
```

```
6.75
```

Some Math Fun

Math operations perform as you'd expect!

Addition

```
print(6 + 15)
```

```
21
```

Multiplication

```
print(6 * 12)
```

```
72
```

Subtraction

```
print(6 - 8)
```

```
-2
```

Division

```
print(45 / 9)
```

```
5.0
```

Returns a float!

Some Math Fun

Works with floats too

Addition

```
print(6.3 + 15.0)
```

```
21.3
```

Multiplication

```
print(6.8 * 12.2)
```

```
82.96
```

Subtraction

```
print(6.5 - 8)
```

```
-1.5
```

Returns a float even with an integer

Division

```
print(46.75 / 2.75)
```

```
17.0
```

Math with Strings?

```
print('Math is' + 'fun')
```

```
Math is fun
```

We have seen this before!

```
print('zig' * 2)
```

```
?
```

Math with Strings?

```
print('Math is' + 'fun')
```

```
21.3
```

We have seen this before!

```
print('zig' * 2)
```

```
zigzig
```

More Math

```
print(7 // 4)
```

```
1
```

Floor Division

Divides and removes the decimal part (Rounding down)

```
print(7 % 2)
```

```
1
```

Modulus

Divides and returns the remainder

```
print(2 ** 5)
```

```
32
```

Exponent

Takes the power
eg. 2^5

Tips to save time

```
x = x + 2
x = x - 4
x = x * 5
x = x / 2
```

→

```
x += 2
x -= 4
x *= 5
x /= 2
```

Does it work for the other operators we learned?
Try it!

Try it!

You are the manager at a bustling Chelsea restaurant and one of your key responsibilities is to oversee reservations to allow the most number of guests to be seated every night.

A large reservation for 100 guests just came in and you must determine the best table configuration to accommodate this group of guests. Write Python code to answer the following questions

- The tables can seat 7 guests. How many tables will need to be reserved?
- How many guests in this group will not be seated at a full table?
- You expect each table to spend around $500 if they are at a full table and $50 per person if they are not at a full table. What is the expected revenue from this reservation?
- If there is 22% gratuity, what is the group's total expenditure?

Solution

```
num_tables = 100 // 7
print(num_tables)
```

```
num_guests = 100 // 7
print(num_guests)
```

```
revenue = 500*num_tables + 50*num_guests
expenditure = revenue + 0.22*revenue
print(revenue)
print(expenditure)
```

Tip: Variables are great for storing information

Deep Dive: Lists

What is a list?

Definition

A data type for storing values together

How to Create: Use []

```
list_of_numbers = [1, 2, 3, 4, 5]
```

What can be in a list?

`[1, 2, 3, 4, 5]` → Numbers (Integer/ Float)

`['a', 'b', 'c']` → Strings

`[True]` → Booleans

`[1,'a', 'b', False, 32]` → Lists can contain elements of different types!

`[]` → Lists can be created empty

`[[1, 2], [4, 5]]` → Lists can contain lists!

Each element is separated by a comma

List Indexing

Bracket Notation

A character at index ***i*** of a list ***L*** can be accessed using bracket notation: ***L[i]***

```
letters = ['a', 'b', 'c', 'd']
letters[0] = ?
```

Indexing lists is similar to indexing strings, except with elements instead of characters

List Slicing

List indexing uses bracket notation with 1 index given, eg: `L[2]`
List slicing uses bracket notation with 2 indexes given, eg: `L[2:4]`

List Slicing

The resulting substring contains all elements starting from the first index given, up to but ***NOT*** including, the last index given

Sound familiar? List slicing is similar to strings too!

List Slicing

```
letters = ['a', 'b', 'c', 'd']
letters[:2] = ?
letters[-3:] = ?
letters[1:] = ?
```

More Overlappings with Strings

print(list_name)

Print a list

len(list_name)

Find the length of a list, ie. How many elements are in it?

list_name.index(element)

Returns the index at which the given element **first** appears in the list

List Methods

`.append()`	Add an element to the back of the list
`.extend()`	Join a list to the back of another list
`.insert()`	Add an element to a specific position
`.pop()`	Remove the last element and return it
`.sort()`	Sort the elements inside the list
`[]`	Change the element at a specific position

List Methods

Recall: Methods typically follow the syntax **`noun.verb()`**

.append() Add an element to the back of the list

```
list_name.append(element)
```

```
letters = ['a', 'b', 'c', 'd']
letters.append(True)
```

Nothing is returned! Why?

List Methods

```
letters = ['a', 'b', 'c', 'd']
letters.append(True)
print(letters)
```

```
['a', 'b', 'c', 'd', True]
```

Now something is returned! Why?

List Methods

List methods work on the list and do not return anything, with the exception of `pop()`

If you want to view the list, you have to explicitly call it after running the list method

List Methods

.extend() Join a list to the back of another list

```
list1.extend(list2)
```

```
letters = ['a', 'b', 'c', 'd']
numbers = [1, 2, 3]
letters.extend(numbers)
print(letters)
```

```
['a', 'b', 'c', 'd', 1, 2, 3]
```

List Methods

.insert() Add an element to a specific position

```
list_name.insert(index_desired_in_final_product, element)
```

```
letters = ['a', 'b', 'c', 'd']
letters.insert(-1, 'z')
print(letters)
```

```
['a', 'b', 'c', 'd', 'z']
```

List Methods

[] Change the element at a specific position

```
list_name[index] = new_element
```

```
letters = ['a', 'b', 'c', 'd']
letters[0] = 100
print(letters)
```

```
[100, 'b', 'c', 'd']
```

List Methods

.sort() Sort the elements in a list

```
list_name.sort(reverse=True/False)
```

```
numbers = [20, 48, 3]
```

```
numbers.sort(reverse=True)
print(numbers)
```

```
[48, 20, 3]
```

```
numbers.sort(reverse=False)
print(numbers)
```

```
[3, 20, 48]
```

List Methods

```
words = ['this', 'may', 'be', 'cool']
words.sort(reverse=True)
print(words)
```

```
['this', 'may', 'cool', 'be']
```

List Methods

```
words = ['this', 'may', 'be', 'cool' '!', '2']
words.sort(reverse=False)
```

```
['!', '2', 'be', 'cool', 'may', 'this']
```

symbols < numbers < letters!

List Methods

```
words = ['this', 'may', 'be', 'cool' '!', 2]
words.sort(reverse=False)
```

```
TypeError: '<' not supported between
instances of 'int' and 'str'
```

List Methods

.pop() Remove the last element and <u>return it</u>

```
list_name.pop()
```

```
letters = ['a', 'b', 'c', 'd']
last_elem = letters.pop()
print(last_elem)
```

```
'd'
```

```
print(letters)
```

```
['a', 'b', 'c']
```

List Methods

.pop() Remove the last element and <u>return it</u>

```
list_name.pop()
```

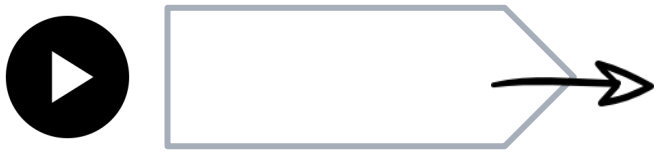
```
last_elem = letters.pop()
```

`pop()` returns something, so we can assign a variable name to the thing returned and use this variable later (eg. in a `print()` statement)

What returns something and what doesn't?

What if we try to assign a variable name to the other list method calls?

```
var = letters.append(6)
print(var)
```



There is nothing assigned to `var`, because all other list methods return nothing when called!

Try it!

You are given a list of profits, each element corresponding to profits from a different consumer group.

```
profits = [180, 230, 567, 4900]
```

Your company has just started targeting a new group of consumers, making a profit of $90 from them (sounds like this targeting strategy isn't great!).

1) Update the company's profit list to reflect this.
2) Return the top 2 highest profits made in a list of the following format: [highest_profit, second_highest_profit]
   Note: We don't care about the other profit figures, only return the top 2!

Try it!

Your company started targeting even more consumer groups, and your profit list has expanded to include them:

```
profits = [180, 230, 567, 4900, 2940, 249,
8341, 9234, 923, 8954, 8123, 7159, 8932, 59,
1001, 9230, 294, 2465, 9845, 2865, 278, 135]
```

You have another list depicting the names of all the consumer groups corresponding to the profits list, ie: The consumer group at index *i* of the names list generated a profit at index *i* of the profits list.

Try it!

```
consumer_group_names = ['under_eighteen', 'over_twenty', 'elderly',
'married', 'single', 'divorced', 'with_kids', 'without_kids', 'luxury',
'budget', 'value-for-money', 'north', 'south', 'east', 'west', 'domestic',
'international', 'working', 'retired', 'student', 'unemployed', 'male',
'female']
```

Find the name of consumer group generating the most profits.

Remember: the consumer at index 2 in `consumer_group_names` (elderly) generated the profit at index 2 in `profits` on the previous slide (567)

Try it!

Your company has re-invented the way it stores its data, and your profits list has become:

```
profits = [['under_eighteen', 243],
['over_twenty', 937], ['elderly', 824],
['married', 123], ['single', 343]]
```

Return the profits made in the over_twenty consumer group.

Advanced Concepts

```
a = [1,2]
b = a
b[1] = 5
print(a)
```

```
a = [1,2]
b = a
b.append(3)
print(a)
```

solution: use b = list(a)

```
a = [1,2]
b = list(a)
b[1] = 5
print(a)
```

Pointer concept: b=a not copying list over, just copying pointer to that list, point to the same list in memory

Deep Dive: Tuples

What is a tuple?

Definition

A data type for storing values together

How to Create: Use ()

```
tuple_of_numbers = (1, 2, 3, 4, 5)
```

Hold on - doesn't this look a list?

Difference

Immutable - unable to change the elements of a tuple

```
letters = ('a', 'b', 'c', 'd')
letters[0] = 'e'
```

Tuple indexing works the same as list indexing!

```
TypeError: 'tuple' object does not support item assignment
```

Behaves almost like a list

Key Similarities

- Can contain elements of different types
- Indexing, slicing

```
my_tuple = (1, 'a', 'b', False, 32, [5,6])
my_tuple[2] = 'b'
my_tuple[4:6] = (32, [5,6])
```

Appendix

Mutable elements within the tuple

```
my_tuple = ([False, 'a'], [5,6])
my_tuple[1] = ?
```

Mutable elements within the tuple

```
my_tuple = ([False, 'a'], [5,6])
my_tuple[1]  ([5,6])
my_tuple[1][1] = 7
```

Do you think there will be an error?

Mutable elements within the tuple

```
my_tuple = ([False, 'a'], [5,6])
my_tuple[1] = ([5,6])
my_tuple[1][1] = 7
```

Do you think there will be an error?

```
print(my_tuple)
([False, 'a'], [5,7])
```

Elements within a mutable data type can be changed

Tuple Methods

`.index()` Returns index of element

`.count()` Returns number of occurrences of element

Tuple Methods

.index() Returns the index at which the given element **first** appears in the list

```
tuple_name.index(element)
```

```
my_tuple = (1, 'a', 'b', False, 32,
 [5,6])
my_tuple.index(32) = ?
```

Tuple Methods

.index() Returns the index at which the given element **first** appears in the list

```
tuple_name.index(element)
```

```
my_tuple = (1, 'a', 'b', False, 32,
 [5,6], 32)
my_tuple.index(32) = 4
```

Tuple Methods

.count() Returns number of occurrences of element

```
tuple_name.count(element)
```

```
my_tuple = (1, 'a', 'b', False, 32,
 [5,6], 32)
my_tuple.count(32) = ?
```

Tuple Methods

.count() Returns number of occurrences of element

```
tuple_name.count(element)
```

```
my_tuple = (1, 'a', 'b', False, 32,
 [5,6], 32)
my_tuple.count(32) = 2
```