TECHNION

Azrieli Continuing Education and
External Studies Division

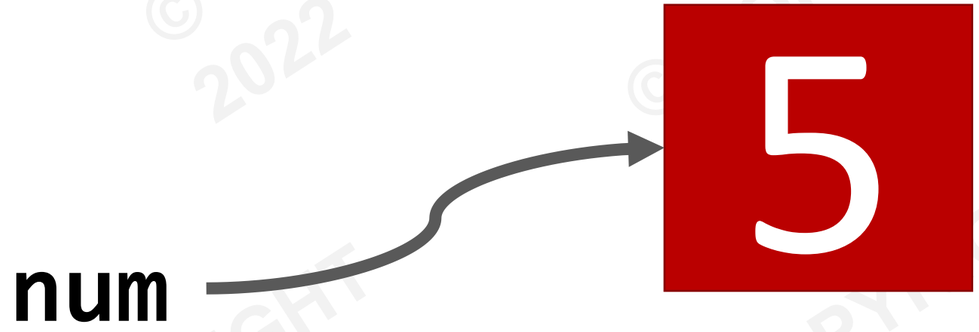
Module 5.3.2: Tuples and sets

Advanced Data Structures

How Variables Work

Let’s save a number:

```
num = 5
```



What just happened?

Python saved the number 5 someplace in the computer’s memory,

then gave us back a ***pointer*** named *num*.

This pointer has a name (*num*) and knows where to look for the value (5) in the computer’s memory.

And a list?

What if we do the same thing with a list?

```
numbers = [8, 6, 4, 2]
```

The same thing happens!

The list is created in memory, and a pointer named *numbers* is given back to us.

This pointer knows where to find the list data in the computer's memory.

But check out the difference…

```
In [1]: num = 5

In [2]: second_num = num

In [3]: second_num = 100

In [4]: num
Out[4]: 5
```

Create a variable

Make a copy

Change the copy

What happens to the original?

It did not change.

And lists?

```
In [5]: numbers = [8, 6, 4, 2]

In [6]: other_numbers = numbers

In [7]: other_numbers[1] = 100

In [8]: numbers
Out[8]: [8, 100, 4, 2]
```

Create a variable

Make a copy

Change the copy

What happened to the original?

It changed!

Wait, what??

```
In [1]: num = 5

In [2]: second_num = num

In [3]: second_num = 100

In [4]: num
Out[4]: 5
```

Create a variable

Make a copy

Change the copy

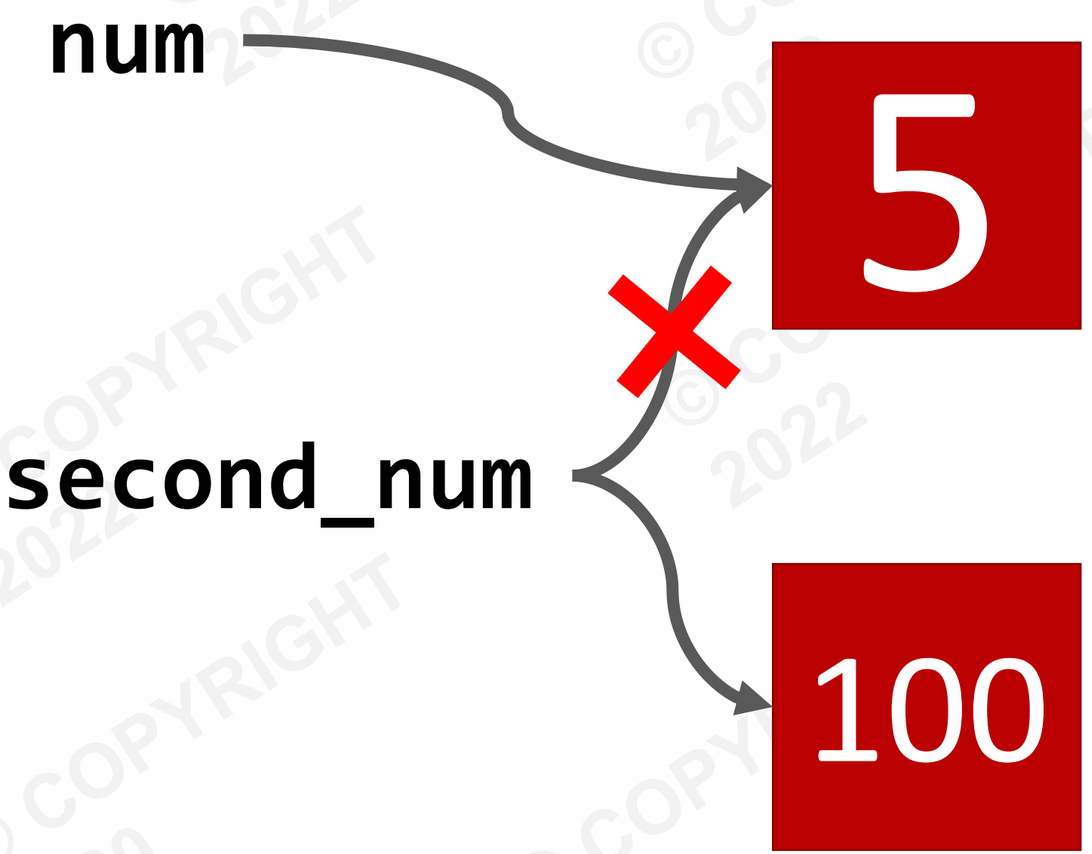
```
In [5]: numbers = [8, 6, 4, 2]

In [6]: other_numbers = numbers

In [7]: other_numbers[1] = 100

In [8]: numbers
Out[8]: [8, 100, 4, 2]
```

The list changes.

The integer does not.

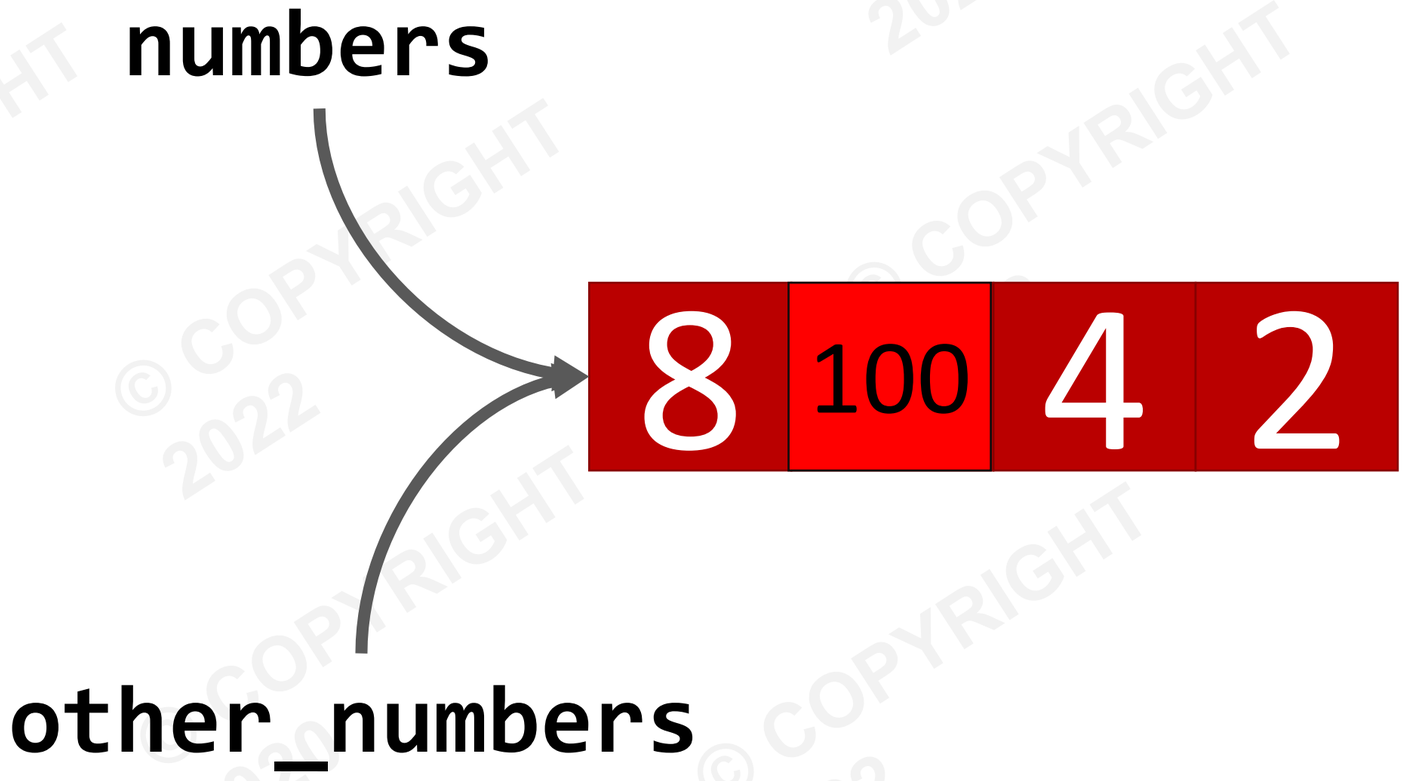
Let us see what is happening behind the scenes…

- num = 5
- second_num = num
- second_num = 100



Behind the scenes with lists…

- numbers = [8, 6, 4, 2]
- other_numbers = numbers
- other_numbers[1] = 100



There are two categories of types

All types in Python are either ***mutable*** or ***immutable***.

Mutable types can be changed!

Immutable types cannot be changed.

Mutable vs. Immutable Types

Mutable	Immutable
list	int
	float
	str
	tuple
	bool

Let us check!

```
In [9]: word = 'Cyberrrrr'

In [10]: word[4] = 'a'
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-10-8d14de5f462b> in <module>
----> 1 word[4] = 'a'

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

- Why did this happen?
- Because strings are immutable!

Let’s check!

```
In [17]: letters = [1, 2, 3, 4, 5]

In [18]: letters[0] = 9999

In [19]: letters
Out[19]: [9999, 2, 3, 4, 5]
```

- Why did this happen?
- Because lists are mutable!

Tuples

Tuples

- ***Tuples*** are very similar to lists.
  They differ only in that tuples cannot be modified.
- To define a tuple, you use regular parentheses – () – instead of square brackets.

```
>>> names = ('Andy', 'Simon', 'Josh')
>>> type(names)
<class 'tuple'>
>>> names[1] = 'Alex'
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    names[1] = 'Alex'
TypeError: 'tuple' object does not support item assignment
>>>
```

List vs Tuple

	• Lists	Tuples
Define	Square Brackets - [ ]	Regular Brackets - Parentheses - ( )
Attributes	Add data Remove data Change data	Cannot be changed
Size		
Runtime		

List vs Tuple - Size

Lists occupies more memory than tuples!
In order to measure and demonstrate the size difference, we will have to import to sys module!
The sys module consists of the *getsizeof*() function, which returns the size of an object in bytes!

List vs Tuple - Time

Tuples can be made more quickly than lists!
In order to measure the time difference, we will have to import the *timeit* module.
The *timeit* module consists of the *timeit*() function, which returns how long it takes to finish a process!

Packing

- Tuples are very useful in Python. In fact, they are so embedded in the language that you don’t even need parentheses to define them!
- This is called ***packing***.

```
>>> nums = 1, 2, 3
>>> nums
(1, 2, 3)
>>> type(nums)
<class 'tuple'>
```

Unpacking

- Unpacking is the exact opposite of packing – it takes a tuple and divides it into different variables.

```
>>> nums = (111, 222, 333)
>>> a, b, c = nums
>>> a
111
>>> b
222
>>> c
333
>>>
```

Packing - Unpacking

- This is a very, *very* useful technique in Python – this uses both packing and unpacking to put multiple values into multiple variables.

- This can be used to switch between two variables without use of a third, or to do more complex calculations.

```
>>> a, b = 5, 8
>>> a
5
>>> b
8
>>> a, b = b, a + b
>>> a
8
>>> b
13
```

Tuples Summary

- The Tuple Type
- Tuples VS Lists
- Packing
- Unpacking

Sets

Creating a Set

- Sets are a new sequence type, created using curly brackets:

```
In [1]: letters = {'a', 'b', 'c', 'd', 'e'}

In [2]: letters
Out[2]: {'a', 'b', 'c', 'd', 'e'}

In [3]: type(letters)
Out[3]: set
```

Set Qualities

- Sets have two main qualities:
- Sets do not have any specific **order**.
  - They do not even support indexing!
- Sets hold only **distinct** values.
  - This means that there are no duplicate values!

```
In [5]: letters = {'a', 'a', 'b', 'a', 'b', 'c', 'a'}

In [6]: letters
Out[6]: {'a', 'b', 'c'}
```

Creating a Distinct List

```
In [7]: nums = [1, 2, 1, 5, 5, 4, 4, 4, 3, 2, 1, 5, 1]

In [8]: nums
Out[8]: [1, 2, 1, 5, 5, 4, 4, 4, 3, 2, 1, 5, 1]

In [9]: nums_set = set(nums)

In [10]: nums_set
Out[10]: {1, 2, 3, 4, 5}

In [11]: new_nums = list(set(nums))

In [12]: new_nums
Out[12]: [1, 2, 3, 4, 5]
```

Problem:

The result can be any order!

Using the in Keyword

- Just like in lists, tuples, and dicts – using the ***in*** keyword can check if an item is in the set.

```
In [24]: capitals = {'Doha', 'Amman', 'Baghdad'}

In [25]: 'Doha' in capitals
Out[25]: True
```

Set Methods and Operators

Method Name	Explanation	Example
add	Adds a new value to the set	`{1, 2, 3}.add(4)`
remove	Removes a value from the set	`{1, 2, 3}.remove(2)`
intersection	Returns a set with the items in the set that appear in the second set.	`{1, 3, 5}.intersection({3, 4, 5})` `>>> {3, 5}`
set1 – set2	Returns a set with the items in set1 that don't appear in set2.	`{1, 3, 5} - {3, 4, 5}` `>>> {1}`
issubset	Returns True/False, depending on if all items in the set appear in the second set.	`{0, 4, 7}.issubset(set(range(10)))` `>>> True`

Mutable vs. Immutable Types

Mutable	Immutable
list	int
dict	float
set	str
	tuple
	bool

What did we learn?

- Sets
- Qualities
- Methods