UNSW Business School

COMM5007 Coding for Business

Lecture 3
Python Fundamentals Part 3

Jan 2024

Copyright

- There are some file-sharing websites that specialise in buying and selling academic work to and from university students.

- If you upload your original work to these websites, and if another student downloads and presents it as their own either wholly or partially, you might be found guilty of collusion — even years after graduation.

- These file-sharing websites may also accept purchase of course materials, such as copies of lecture slides and tutorial handouts. By law, the copyright on course materials, developed by UNSW staff in the course of their employment, belongs to UNSW. It constitutes copyright infringement, if not academic misconduct, to trade these materials.

Country

- UNSW Business School acknowledges the Bidjigal (Kensington campus) and Gadigal (City campus) the traditional custodians of the lands where each campus is located.

- We acknowledge all Aboriginal and Torres Strait Islander Elders, past and present and their communities who have shared and practiced their teachings over thousands of years including business practices.

- We recognize Aboriginal and Torres Strait Islander people's ongoing leadership and contributions, including to business, education and industry.

UNSW Business School. (2022, August 18). *Acknowledgement of Country* [online video]. Retrieved from https://vimeo.com/369229957/d995d8087f

At UNSW you are free to...

Respectfully disagree about anything

Express different opinions

Write your beliefs

Show your beliefs

Leave any club or organisation

It's not acceptable to...

Attempt to censor opinions

Use hate speech

Make threats or instil fear

Make false accusations

Access or share others private information without consent

We are here to help...

Tell a teacher

Tell UNSW Psychology and Wellness

Report to UNSW Complaints

Report to UNSW Security

Report a crime to police

Find out more

Reminder on Quiz 1

- Quiz 1 **at the beginning** of Lab 3 on Wed, Jan 10 and Thurs, Jan 11
- What to bring
  - **2B pencil and eraser**
  - Any printed material (Open book, but not open laptop)
  - Student card (tutor will verify)
- Each lab will use different questions, but at the same difficulty level
- The content in lecture 1 and 2 is covered
- No collaboration
- 7 Multiple choice questions; 15 minutes; 7.5% of final grade
- Only 1 answer for each question; no negative mark for wrong answers

Individual and Team Assignment

- 10%: One (1) individual assignment
  - Turnitin submission of .docx file
  - Publish on Wed, 10 Jan 2024
  - Submission by Fri, 19 Jan 2024, 14:59

- 30% team assignment
  - Lab 8: Team Assignment Presentation in person on campus (hence Monday tutorials in week 5 are in person)
  - Publish on Fri, 19 Jan 2024
  - Submission: .ipynb, .pptx by Mon, 29 Jan 2024, 12:59

Outline

- **Tuple**
- List
- Set
- Dictionary

Tuple

- A **tuple** is a sequence of values separated by commas.
  - The values stored in a tuple can be any type, and they are indexed by integers.
  - Tuples are used to store **multiple items** in **a single variable**.

```
my_tuple = ("country music", 5, 3.65)
                # music style, music typeID, avg length
my_tuple
```

- Check the type of the tuple

```
type(my_tuple)
```

Indexing tuple elements

- A tuple may have many elements
- Each element can be accessed via an index
- Relationship between the index and the items in the tuple

```
my_tuple = ("country music", 5, 3.65)
```

Index	Index	element
0	-3	"country music"
1	-2	5
2	-1	3.65

```
print(my_tuple[0])
print(my_tuple[1])
print(my_tuple[2])

print(my_tuple[-3])
print(my_tuple[-2])
print(my_tuple[-1])
```

Slicing tuples

- We can slice a tuple to obtain new tuples with the corresponding elements.

- We can obtain the first N elements of a tuple

```
tuple1 = (“country music”, 5, 3.65, 25010, "yes")
tuple1[0:3] # first 3 elements
```

- We can obtain the last N elements of a tuple

```
tuple1[-3:] # last 3 elements
```

Tuples are immutable

We cannot change the elements of the tuple or the tuple itself once it's created.

```
# an example of immutable
Creating a tuple
my_tuple = (1, 2, 3, 4, 5)

# Trying to change a value in the tuple
my_tuple[0] = 10  # what happens?
```

We can concatenate tuples by using the + sign

```
tuple1 = ("country music", 5, 3.65)
tuple2 = tuple1 + (25010, "yes") # total music count, availability
tuple2
```

Index	Index	element
0	-5	**"country music"**
1	-4	**5**
2	-3	**3.65**
3	-2	**25010**
4	-1	**"yes"**

Nested tuple

- A tuple can contain another tuple as well as other data types. This process is called “nesting”.
- Consider the following tuple with nesting:

```
tuple_nested = ( ("country music", "country and western") , 5, 3.65)
```

- In a nested tuple, an element can be a tuple.

Index	Element
0	**("country music", "country and western")**
1	**5**
2	**3.65**

Accessing the second level elements in a nested tuple

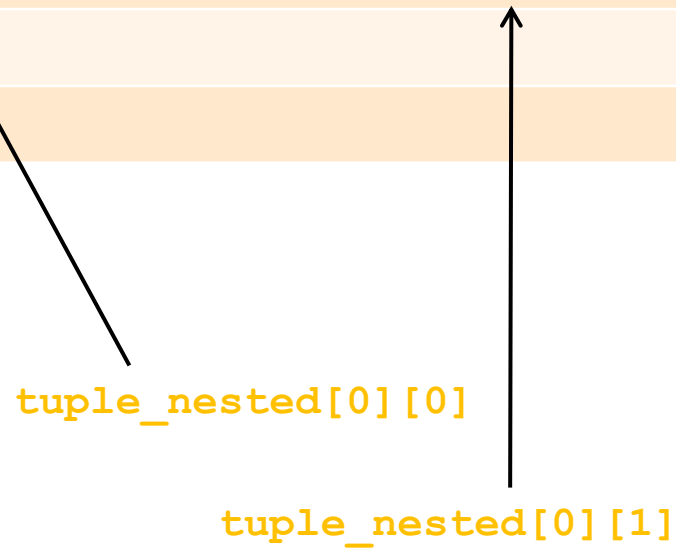
We can use the second index to access the 2nd level elements.

```
tuple_nested = ( ("country music", "country and western") , 10, 3.5)
```

Index	element
0	**("country music", "country and western")**
1	**10**
2	**3.5**

```
# Print element on each index, including nest indexes
print("tuple_nested[0] is:  ",   tuple_nested[0])
print("tuple_nested[0][0] is:  ",   tuple_nested[0][0])
print("tuple_nested[0][1] is:  ",   tuple_nested[0][1])
```

```
tuple_nested[0] is:   ('country music', 'country and western')
tuple_nested[0][0] is:   country music
tuple_nested[0][1] is:   country and western
```



Example: Employee Record

Imagine a human resources system where employee records are stored. Each employee record might include the employee ID, name, role, and hire date.

```
# Defining an employee record as a tuple
employee_record = (101, "John Doe", "Software Engineer", "2023-09-26")

# Accessing elements in the tuple
employee_id = employee_record[0]  # 101
employee_name = employee_record[1]  # "John Doe"
employee_role = employee_record[2]  # "Software Engineer"
hire_date = employee_record[3]  # "2023-09-26"
```

Summary

- A tuple is used to store multiple data items in a single variable.
- A tuple is written with **round** brackets **( )**.
- A tuple is a collection of data items which is **immutable**.

	Mutable	Ordered	Indexing	Duplicate Data
Tuple	✗	✓	✓	✓

Outline

- Tuple
- **List**
- Set
- Dictionary

List

- A list is a collection of elements or items (e.g., strings, numbers) enclosed by a square bracket with each element separated by a comma

```
months = ['Jan', 'Feb', 'Mar’]
```

- A list can contain other Python objects, e.g., a tuple

```
mixed = ['spam', 2.0, 5, (10, 20)]
```

- A list can be empty

```
mixed = []
```

- Print out the list directly will print each element one by one.

```
months = ['Jan', 'Feb', 'Mar']
print(months) # print the list directly
```

Building a new list can be done by two methods

- Method 1: use built-in function `list()`

```
fruits = list()
```

- Method 2: use square brackets

```
fruits = [] # more common
```

Positive indexing in a List

- The address of each element within a list is called an **index**. An index is used to access and refer to items within a list.
- Elements are indexed in list

```
months = ['Jan', 'Feb', 'Mar']
print(months[0])
print(months[1])
print(months[2])
```

Index	element
0	‘Jan’
1	‘Feb’
2	‘Mar’

Negative indexing in a List

The index starts from -1 at the end, and decreases as we go towards the left

Index	element
0 (-3)	‘Jan’
1 (-2)	‘Feb’
2 (-1)	‘Mar’

```
months = ['Jan', 'Feb', 'Mar']
print(months[-3])
print(months[-2])
print(months[-1])
```

List is mutable/changeable, unlike a tuple

- We can **change the items in a list or reassign an item in a list** after the list is formed
- When the index appears on the left side of an assignment, it identifies the element of the list that will be changed

```
days = ['Mon', 'Tue', 'Wed']
days[0] = 'Sun'
print(days)
```

Concatenating and slicing a list

```
first = ['Mon','Tue', 'Wed']
second = ['Thu', 'Fri']
third = ['Sat', 'Sun']

days_of_week = first + second + third   #concatenating
print(days_of_week)
print(len(days_of_week))
print(days_of_week[0:3])  #slicing
```

Note: [0:3] means the index starts with 0, and the ending index is up to but not including index 3

Exercise

Build a list with the following four elements

1, hello, [1,2,3,4] and True

```
my_list = [1, 'hello', [1,2,3,4], True]
```

What does the below retrieve?

```
print(my_list[2][3])
print(my_list[1][2])
```

Popular built-in functions for list operations: `append()`, `sort()`


```
fruits = list() # or fruits = [ ]
fruits.append('apple')
#append() can add ONE element to the end of the list

fruits.append('orange')
fruits.append('banana')
print(fruits)

fruits.sort()
print(fruits)
print(len(fruits))
```

Popular built-in functions for list operations: `insert()`, `extend()`

```
fruits = ['apple', 'orange', 'banana’]
fruits.insert(1, 'peach')
print(fruits)

fruits = ['apple', 'orange', 'banana’]
fruits.extend(['peach’, ‘pear’])
print(fruits)
```

Popular built-in functions for string operations: `.split()`

We can convert a string (a sentence with spaces between words) into a list of words

```
desc = 'This course introduces various techniques'
word_in_desc = desc.split()
print(word_in_desc)
```

Summary of List properties

	Mutable	Ordered	Indexing	Duplicate Data
Tuple	✗	✓	✓	✓
List	✓	✓	✓	✓

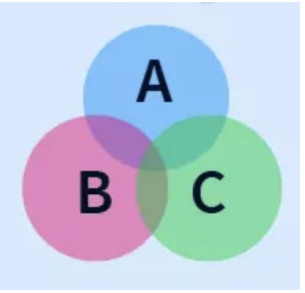
Key differences: Tuple vs. List

List	Tuple
Enclosed by square brackets [ ]	Enclosed by parentheses ( )
Mutable - Elements can be modified - We can add items to a list - We can remove items in a list	Immutable - Elements cannot be modified - We cannot add items to a tuple - We cannot remove items in a tuple
More functions can be performed on a list	Less functions can be performed on a tuple
Less memory efficient	More memory efficient

Outline

- Tuple
- List
- **Set**
- Dictionary

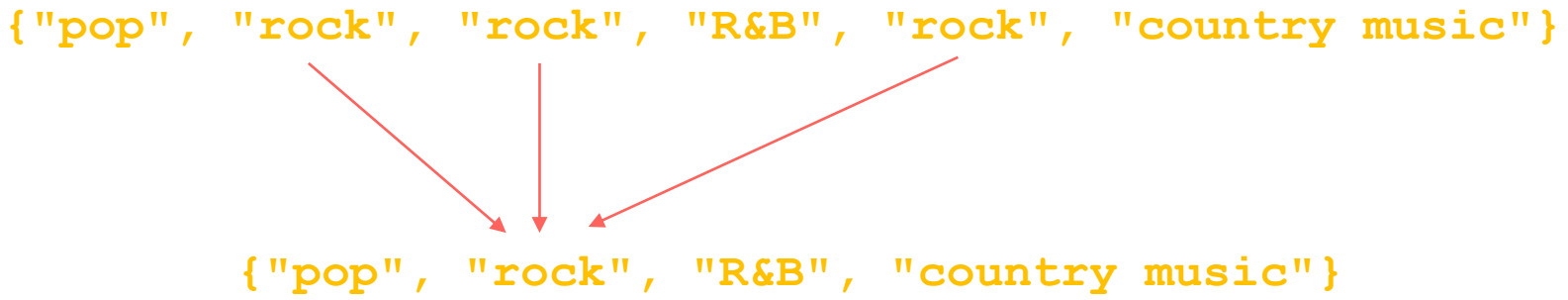
Set



- A set is a collection of **unordered unique** objects in Python. You can denote a set with a pair of curly brackets **{ }**.
  - Objects can be of different data types.
- Python will **automatically remove duplicate items** in a set when you create it.

```
set1 = {"pop", "rock", "rock", "R&B", "rock", "country music"}
set1
```

- What really happened?



Create a set from a list

```
album_list = [ "pop", "rock", "rock", "R&B", "rock" ]
album_set = set(album_list) #set() converts a list to a set
album_set
```

What is the printout?

```
{'R&B', 'pop', 'rock'}
```

Set Operations

- `add()` method: add an element to a set
- `remove()` method: remove an item from a set
- `in` command: verify if an element is in the set

```
A = set(["Yesterday once more", "Five hundred miles",
"Paradise"])
print(A)

A.add("My all in all")
print(A)

A.remove("My all in all")
print(A)

"My all in all" in A
```

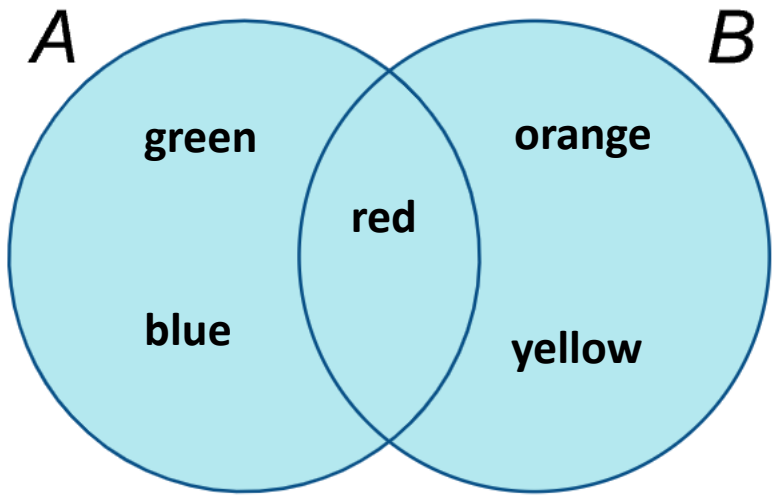
Set Operations: union on two or more sets

- `union()` method
- operator |

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A | B)

# by method
print(A.union(B))
```



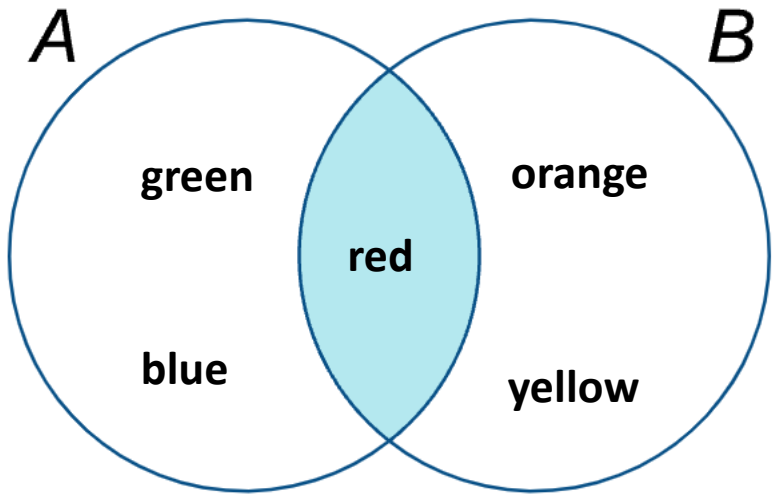
Set Operations: intersection on two or more sets

- `intersection()` method
- operator `&`

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A & B)

# by method
print(A.intersection(B))
```



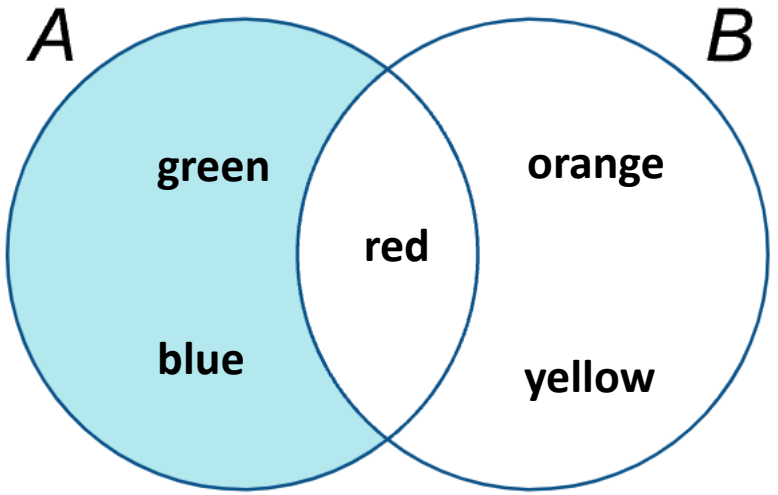
Set Operations: difference between two or more sets

- `difference()` method
- operator `-`

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A - B)

# by method
print(A.difference(B))
```



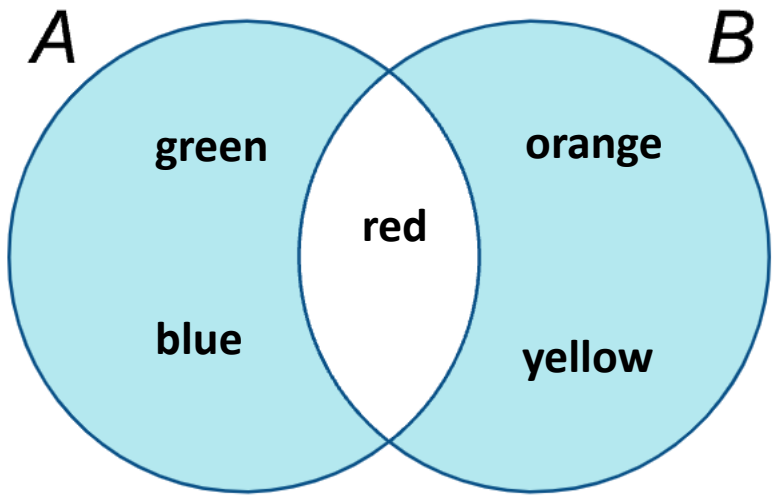
Set Operations: Symmetric difference between two or more sets

- `symmetric_difference()` method
- operator `^`

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A ^ B)

# by method
print(A.symmetric_difference(B)
```



Exercise: Perform set operations

```
A = {0, 2, 4, 6, 8}
B = {1, 2, 3, 4, 5}

# union
print("Union :", A | B)
# intersection
print("Intersection :", A & B)
# difference
print("Difference :", A - B)
# symmetric difference
print("Symmetric difference :", A ^ B)
```

What is the printout?

```
Union : {0, 1, 2, 3, 4, 5, 6, 8}
Intersection : {2, 4}
Difference : {0, 8, 6}
Symmetric difference : {0, 1, 3, 5, 6, 8}
```

Summary of Set properties

	Mutable	Ordered	Indexing	Duplicate Data
Tuple	✗	✓	✓	✓
List	✓	✓	✓	✓
Set	✓	✗	✗	✗

- **Mutable:** Elements can be changed.
- **Unordered:** Items stored in a set aren't kept in any particular order.
- **Unindexed** – You cannot access set items by referring to an index.
- **No duplicate data**

Outline

- Tuple
- List
- Set
- **Dictionary**

Dictionary

- A dictionary is **a mutable, unordered collection** that stores mappings of unique keys to values. The building blocks are **key-value pairs**
- Values can be of any data type
- Values of the Python dictionary may or may not be unique.
- Keys are always **unique/only a single element** within a dictionary, because keys help us locate the data in the dictionary.
- Keys must be **immutable**. Hence, the key can only be **strings, numbers or tuples**.

```
veggie_price = {'green pepper':15, 'tomato':6.9, 'green onion':3.5}
```

What is inside a dictionary?

- A dictionary holds a sequence of elements.
- Each element is a key-value pair

```
veggie_price = {'green pepper':15, 'tomato':6.9, 'green onion':3.5}
```

Key	Value
'green pepper'	15
'tomato'	6.9
'green onion'	3.5

Create a dictionary

- Straightaway type out the key-value pairs in {}

```
veggie_price = {'green pepper':15, 'tomato':6.9, 'green onion':3.5}
print(veggie_price)
```

- Start with an empty dictionary

```
fruit_price = dict()
fruit_price = {} # more common
fruit_price['apple'] = 3.5 # does the key exist previously?
fruit_price['orange'] = 5
fruit_price['blueberry'] = 20
print(fruit_price) #{'apple': 3.5, 'orange': 5, 'blueberry': 20}
```

Extracting data from a dictionary

- We can refer to a key in a dictionary and extract the value for that key, `Dictionary_name['key_name']`

```
veggie_price2 = {'green pepper':15, 'tomato':6.9}
print(veggie_price2 ['tomato’])
# does the key exist?
```

Another way to extract data from a dictionary…

`get()` returns the value given a key as an argument, and does not trigger an error if the key is non-existent

```
veggie_price2 = {'green pepper':15, 'tomato':6.9}
print(veggie_price2)

veggie_price2['potato'] # what happens?

veggie_price2.get('potato') # what happens?
```

Adding and changing the elements in a dictionary

How do we differentiate whether we are adding a new key-value pair, or changing the value of an existing key?

```
veggie_price2 = {'green pepper':15, 'tomato':6.9}
print(veggie_price2)

veggie_price2['green onion'] = 3.5 # add new key-value pair
veggie_price2['tomato'] = 3 # update an existing key's value
print(veggie_price2)
```

Other methods (1/2)

- Get all the keys in dictionary

```
veggie_price2.keys()
```

- Get all the values in dictionary

```
veggie_price2.values()
```

- **`items()`** returns a view object. The view object contains the key-value pairs of the dictionary, as a list of tuples.

```
veggie_price2 = {'green pepper':15, 'tomato':6.9}
veggie_price_tuple = veggie_price2.items()
print(veggie_price_tuple)
```

What's the printout?

```
dict_items([('green pepper', 15), ('tomato', 6.9)])
```

Other methods (2/2)

- Delete elements (key-value pairs) by the key

```
del(veggie_price2['tomato’])
```

- Checking the existence of a key in a dictionary using the ‘**in**’ operator

```
veggie_price2 = {'green pepper':15, 'tomato':6.9}
print('tomato' in veggie_price2.keys())
```

Summary of Dictionary properties

	Mutable	Ordered	Indexing	Duplicate Data
Tuple	✗	✓	✓	✓
List	✓	✓	✓	✓
Set	✓	✗	✗	✗
Dictionary	✓	✓	✓	✗

Built-in Function: `sorted()`

- `sorted()` function takes a given **tuple, list, string, set, or dictionary** as a parameter and return a sorted **list** with the same elements in ascending or descending order.

- E.g., sort a tuple of scores and save the sorted result to **a new list.**

```
scores = (0, 3, 2, 5, 10, 8, 9)
scores_sorted = sorted(scores)
scores_sorted # [0, 2, 3, 5, 8, 9, 10]
```

- How do we sort the tuple in descending order?

```
scores = (0, 3, 2, 5, 10, 8, 9)
scores_sorted = sorted(scores, reverse=True)
scores_sorted # [10, 9, 8, 5, 3, 2, 0]
```

Recap Exercise

- What happens if we sort the dictionary below?
- What are we sorting by?
- What do the sorted result contain? Key-value pairs?

```
name_scores = {'John':15, 'Mary':69, 'Atul': 63}
sorted_list = sorted(name_scores)
sorted_list
```