MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Computer Science Department

COMP8060 – Scientific Programming with Python

Week 4 – Basic Data Structures

Dr. Bruno Andrade

October, 2025

Last week

- Decision Structures:
  - If-elif-else
  - Match-case

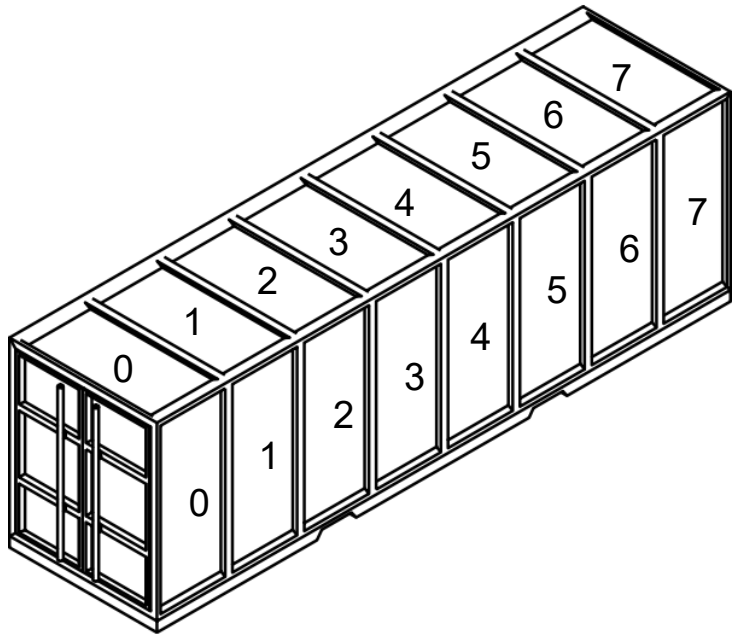
MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

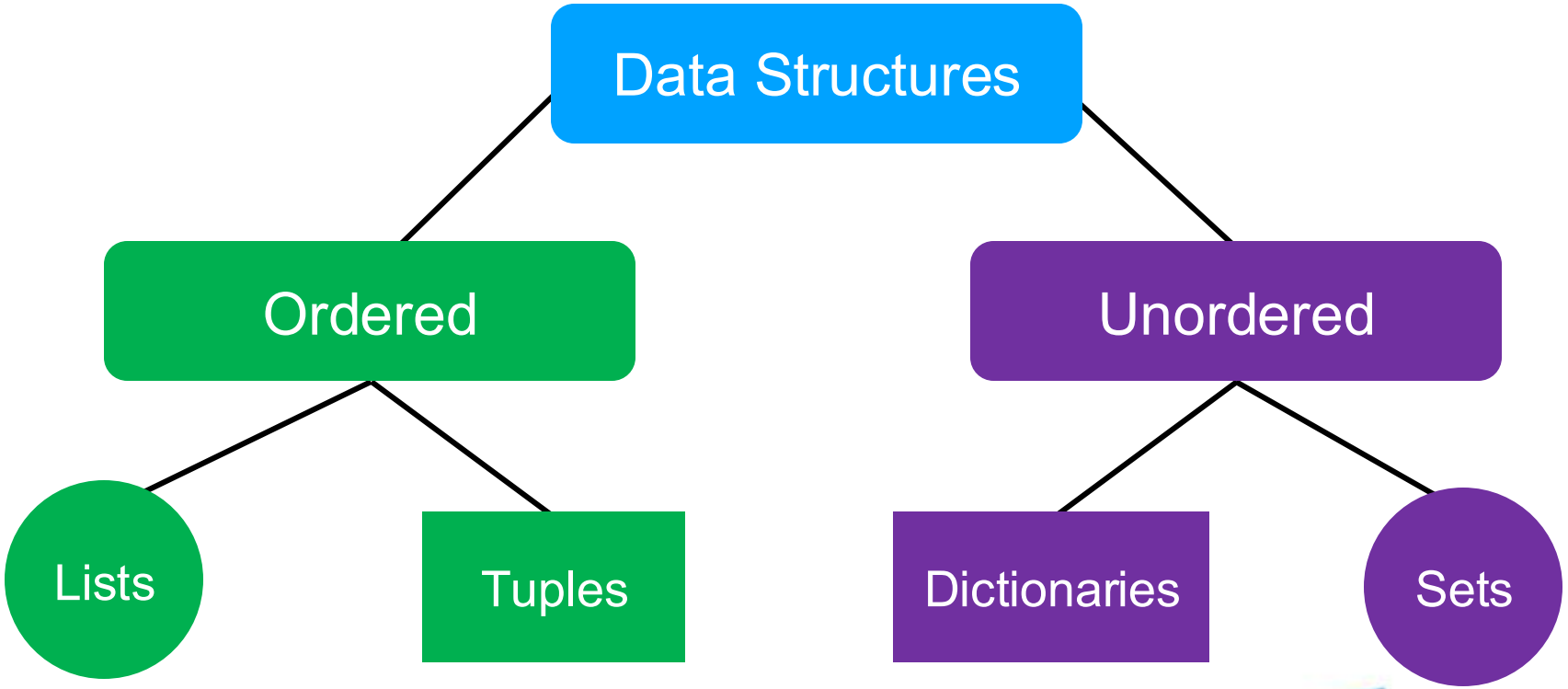
- **Summary:**
  - Data Structures
  - Lists
  - Tuples
  - Dictionary
  - Sets

Data Structure

Data structure

- Python has different types of data Structures.
- They differ on mutability and order.
  - Some are mutable, some not.
  - Some are ordered, some are certainly not
- This week we will see the basic ones,
  Lists, Tuples, Dictionaries and Sets!




Data Structures
Ordered
Unordered
Lists
Tuples
Dictionaries
Sets

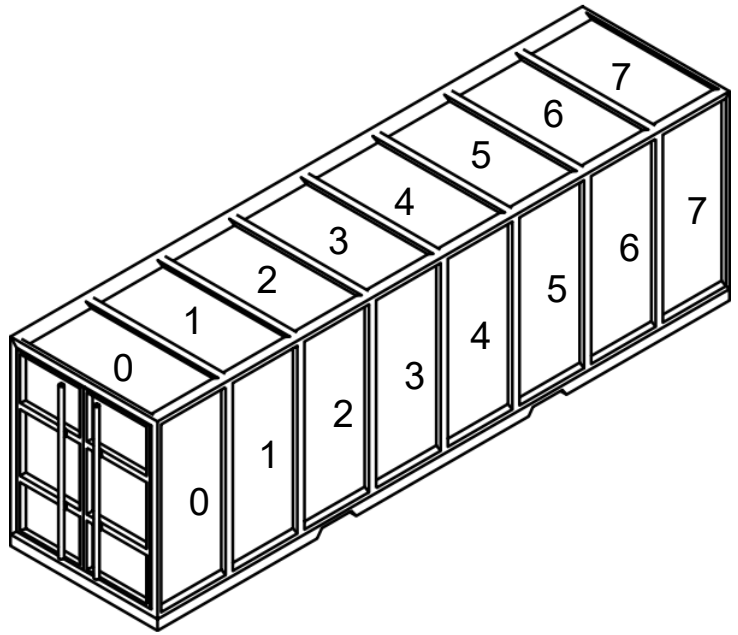
Lists

Introduction to Lists

- A list is a dynamic collection of items in a particular order. Lists are dynamic data structures, meaning that items may be added or removed from them.

Variable Person

List Person[]

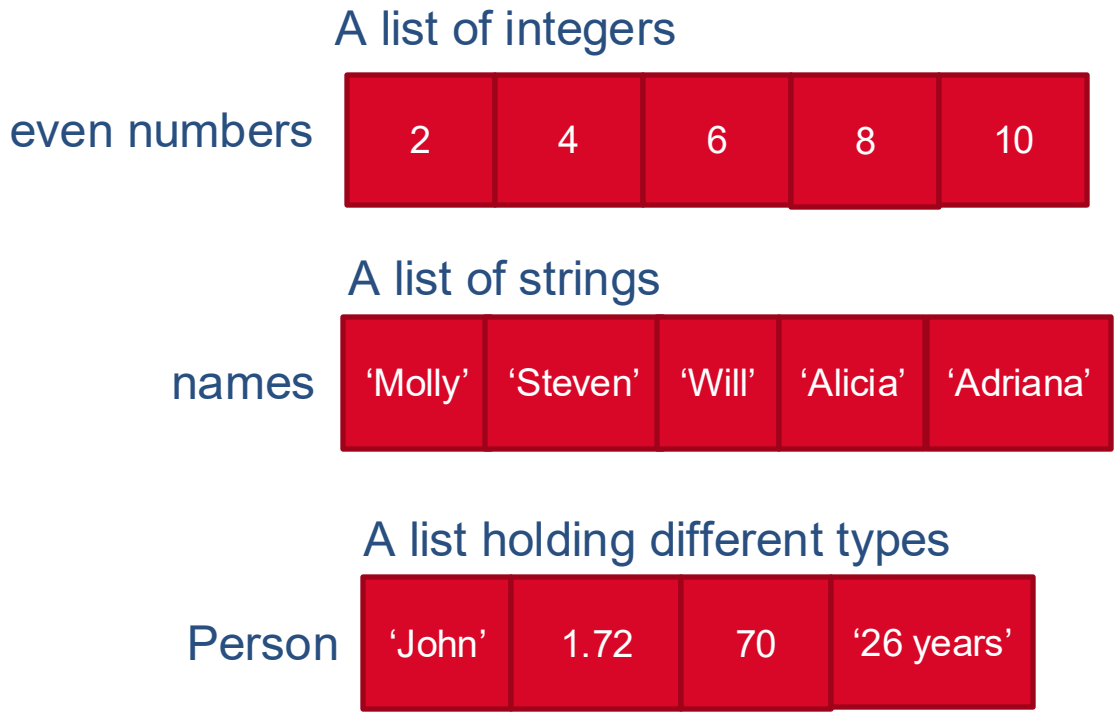


Introduction to Lists

- In Python, a pair of square brackets [] indicate a list, and elements are separated by commas:
  - Format: ***list* = [*item1*, *item2*, etc]**
  - Can hold items of different types

```
emptyList = []
even_numbers = [2, 4, 6, 8, 10]
names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
Person = ['John', 1.72, 70, '26 years']
```

Introduction to Lists (Reference Variables and Mutable Data Structure)



Indexing – Accessing an element at a position within the list

- Index: a number specifying the **position** of an element in a list
  - Enables access to an individual element in a list
  - Index of first element in the list is 0, second element is 1, and n'th element is n-1

Indexing

myList = [10, 20, 30, 40]

- We can access the individual elements of the list with the following statement: **myList[index]**

```
myList=[100,200,300,400,500,600]
print(myList[ ])
print(myList[ ])
```

```
300
100
```

The index of the first data item starts at 0

Indexing

- Negative indexes identify positions relative to the end of the list
- The index -1 identifies the last element, -2 identifies the next to last element, etc.

100	200	300	400	500	600
index -6	index -5	index -4	index -3	index -2	index -1

Indexing

```
myList=[100,200,300,400,500,600]
print(myList[ -2])
print(myList[  -1])
```

```
400
600
```

The len function

- **len** function: returns the length of a sequence such as a list
  - Example: **size = len(my_list)**

- Knowing that the len function will return length of list then how would we use it to access the last element in the list

Returns the number of elements in the list, so the index of last element is `len(list)-1`

The len function

```
myList = [10, 20, 30]
index = 0
while index < len(myList):
    print(myList[index])
    index += 1
```

```
10
20
30
```

```
myList = [10, 20, 30]
index = 0
for x in myList:
    print(x)
```

```
10
20
30
```

Assigning a new value to a list

- Lists are mutable:
  - Their **elements can be changed**

- An expression such as:

```
list[i] = new_value
```

  can be used to assign a new value to a list element where *i* is a valid index within the list
  - Must use a valid index to prevent raising of an `IndexError` exception

Changes Values of a List

```
numbers = [23, 54, 56, 67]
numbers[1] = 43
numbers[3] = 2
numbers[0] = 0
print (numbers)
```

```
[0, 43, 56, 2]
```

The range Function

- The range function returns a range object that can be used to generate a list of numbers.
- We can force the range function to generate the list by passing it to a list function.

```
#will assign the list [0, 1, 2, 3, 4] to the numbers variable.
# Then prints out the contents of the list numbers
numbers = list(range(5))
print (numbers)
```

```
[0, 1, 2, 3, 4]
```

Concatenating Lists

- Concatenate: join two lists together
  - The **+** operator can be used to concatenate two lists (Cannot concatenate a list with another data type, such as a number)

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = list1+list2
# list3 now contains [1, 2, 3, 4, 5, 6]
```

  - The **+=** augmented assignment operator can also be used to concatenate lists.

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1 += list2
#(changes contents of list1 to [1, 2, 3, 4, 5, 6])
```

List Slicing

- Slice: a span of items that are taken from a sequence
  - List slicing format: ***list*[*start* : *end*]**
  - Slicing expressions can include a **step** value and negative indexes relative to end of list

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[2:5]
print(midDays)
```

```
['Tuesday', 'Wednesday', 'Thursday']
```

Slicing Example

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[:2]
print(midDays)
```

```
['Sunday', 'Monday']
```

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[5:]
print(midDays)
```

```
['Friday', 'Saturday']
```

Slicing Example

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[2:7:2]
print(midDays)
```

```
['Tuesday', 'Thursday', 'Saturday']
```

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[-2:]
print(midDays)
```

```
['Friday', 'Saturday']
```

Slicing

- **Invalid indexes do not** cause slicing expressions to **raise an exception**. For example:
  - If the end index specifies a position beyond the end of the list, Python will use the length of the list instead.
  - If the start index is greater than the end index, the slicing expression will return an empty list.

Finding Items in Lists with the `in` Operator

- You can use the in operator to determine whether an item is contained in a list
  - General format: ***item in list***
  - Returns True if the item is in the list, or False if it is not in the list
- Similarly, you can use the **not in** operator to determine whether an item is not in a list

Finding Items in Lists with the `in` Operator

```
# code will output “Found list item” as the value 12 appears
# in the list numbers
numbers = [12,  43, 56]
userNumber = 12

if userNumber in numbers:
    print ("Found list item")
```

```
Found list item
```

List methods

- **append(*item*)**: used to add items to a list – *item* is appended to the end of the existing list

- **index(*item*)**: used to determine where an item is located in a list
  - Returns the index of the first element in the list containing item
  - Raises ValueError exception if *item* not in the list

- **insert(*index, item*)**: used to insert *item* at position *index* in the list

Examples

```
names = ['James', 'Kirk']

print (names)

# Insert a new name at element 0.
names.insert(0, 'Tiberius')

print (names)
```

```
['James', 'Kirk']
['Tiberius', 'James', 'Kirk']
```

```
days = ['Sunday']
days.append("Taco Tuesday")
print(days)
```

```
['Sunday', 'Taco Tuesday']
```

```
days.index('Sunday')
```

```
0
```

List Methods

- **del statement**: removes an element from a specific index in a list
  - General format: del *list*[*i*]

- **Min() and max() functions**: built-in functions that return the item that has the lowest or highest value in a sequence
  - The sequence is passed as an argument

List Methods

- **`sort()`**: used to sort the elements of the list in ascending order
- **`remove(item)`**: removes the first occurrence of *`item`* in the list
- **`reverse()`**: reverses the order of the elements in the list

Using Multiple Lists

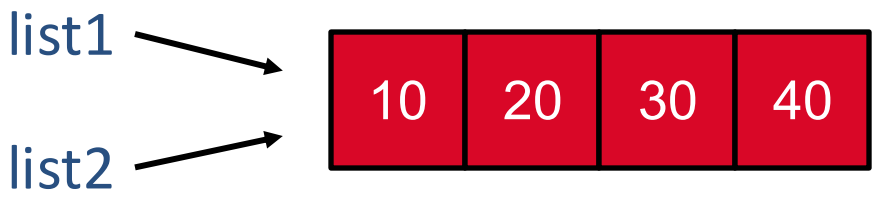
```
list1 = [10, 20, 30, 40]
list2 = list1

# double each value in list 2
for counter in range(len(list2)):
    list2[counter] = list2[counter]*2

print (list1)
print (list2)
```

```
[20, 40, 60, 80]
[20, 40, 60, 80]
```

What is the output of this program?

Copying Lists

- They are pointing to the same list in memory!



- How can we work out this problem?

Copying Lists

- To make a copy of a list you must copy each element of the list
  - Three methods to do this:
    - Create a new empty list and use a `for` loop to add a copy of each element from the original list to the new list
    - Create a new empty list and concatenate the old list to the new empty list
    - Or… We make a copy of the list with the method .copy()!

Examples


```
# Create a list with values.
list1 = [1, 2, 3, 4]


# Create an empty list and concatenate listl to it.
list2 = [] + list1


list2 = []
# Individually copy the elements of listl to list2.
for item in list1:
    list2.append(item)

#Using the copy function
list2 = list1.copy()
```

Accessing the Individual Characters in a String

- Strings behave much like lists, so we can access letters in a similar fashion to a list.
- len(*string*) function can be used to obtain the length of a string
  - Useful to prevent loops from iterating beyond the end of a string

```
city = 'Boston'
index = 0
while index < len(city):
    print (city[index])
    index += 2
```

```
B
s
o
```

```
for x in city:
    print(x)
```

```
B
o
s
t
o
n
```

Split Function

- The split function: returns a list containing the words in the string
  - By default, uses space as separator

```
data = 'John,Doe,1984,4,1,male'

tokens = data.split(',')
firstName = tokens[0]
lastName = tokens[1]
print (firstName, lastName)
```

```
John Doe
```

Tuples

Tuples

- Like lists, a tuple is a collection of items in a particular order. However, tuples are not dynamic, as they cannot be changed!!!

Variable Person

John

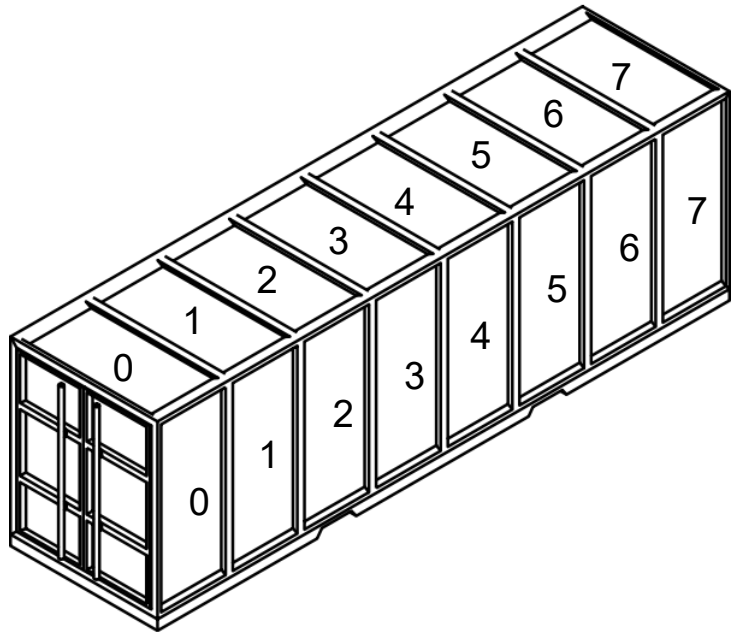
Tuple Person

John

1,72m

70kg

26 years



Tuples

- In Python, a pair of parenthesis () indicate a tupple, and elements are separated by commas:
  - Element: An item in a list
  - Format: ***tuple* = (*item1*, *item2*, etc…)**
  - Can hold items of different types

```
even_numbers = (2, 4, 6, 8, 10)
names = ('Molly', 'Steven', 'Will', 'Alicia', 'Adriana')
Person = ('John', '1,72m', '70kg', '26 years')
print(even_numbers)
```

```
(2, 4, 6, 8, 10)
```

Tuples

- Most methods applied to lists work with Tuples.
- The exception are those used to change the tupple itself.

```
names.remove('Molly')
```

```
---------------------------------------------------------------------
----------
AttributeError                            Traceback (most recent
call last)
/var/folders/9x/hx0451rj4n1fcxsm1m6ch5hx7p3m0h/T/ipykernel_10177/
1071800684.py in <module>
----> 1 names.remove('Molly')

AttributeError: 'tuple' object has no attribute 'remove'
```

Tuples

- The only way to change the value of a tuple is to converting it to a list first (you will be working with a copy though…)

```
names_list = list(names)
print(names_list)
names_list.remove('Molly')
print(names_list)
```

```
['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
['Steven', 'Will', 'Alicia', 'Adriana']
```

- It feels like cheating, but you were never supposed to change a Tuple in the first place ;).

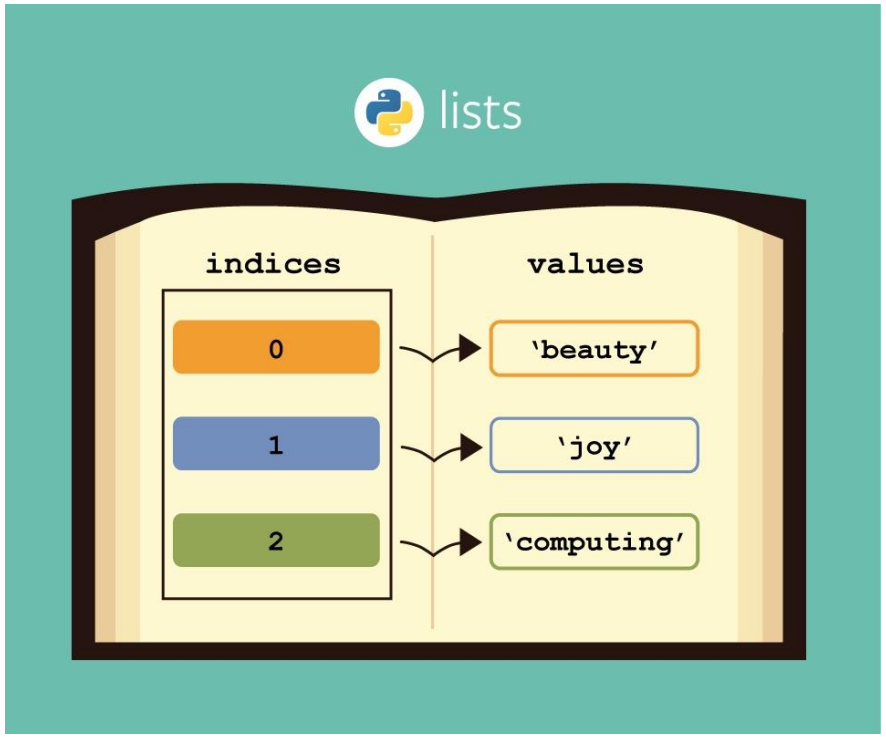
Can you think of a reason to have an immutable list in the first place?

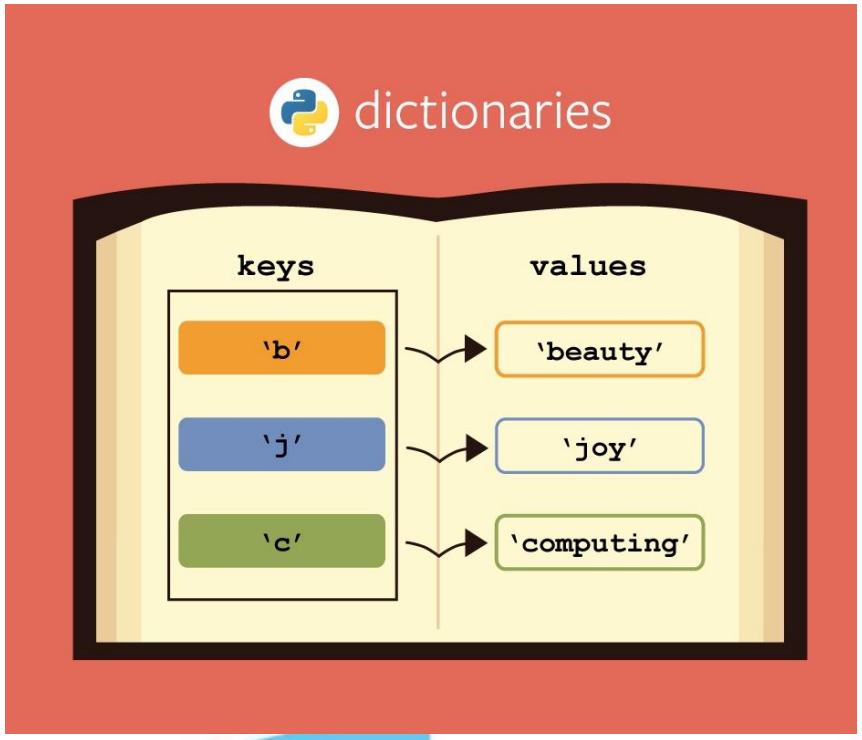
MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Dictionaries

Dictionaries

- Dictionary: object that stores a collection of data





Dictionaries

- Dictionary syntax:
  - Each element consists of a *key* and a *value* pair

```
dictionary = {key1:val1, key2:val2}
```

```
details = {'Jim': 'Engineer', 'John': 'Doctor', 'Kevin':
                                'Chemist'}
```

Dictionaries are way faster than lists or tuples, like, waaaay faster.

Retrieving a Value from a Dictionary - Example

- Elements in a dictionary are unsorted

- General format for retrieving a value from dictionary: *dict[key]*

```
details = {'Jim': 'Engineer', 'John': 'Doctor'
           , 'Kevin': 'Chemist'};
print (details['John'])
```

```
Doctor
```

Retrieving a Value from a Dictionary - Example

- Test whether a key is in a dictionary using the `in` and `not in` operators

```
details = {'Jim': 'Engineer', 'John': 'Doctor'
           , 'Kevin': 'Chemist'};
if 'John' in details:
    print (details['John'])
```

```
Doctor
```

Adding Elements to an Existing Dictionary

- To add a new key-value pair:

  dictionary[key] = value

- **len** function: used to obtain number of elements in a dictionary

  len(dictionary)

```
details = {'Jim': 'Engineer', 'John': 'Doctor'
           , 'Kevin': 'Chemist'}

details['John']= 'Teacher'
print(details)
print(len(details))
```

```
{'Jim': 'Engineer', 'John': 'Teacher', 'Kevin': 'Chemist'}
3
```

Deleting Elements From an Existing Dictionary

- To delete a key-value pair:

```
del dictionary[key]
```

```
details = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
print(details)
print('')
del details['Name']; # remove entry with key 'Name'
print(details)
```

```
{'Name': 'Zara', 'Age': 7, 'Class': 'First'}

{'Age': 7, 'Class': 'First'}
```

Properties of Dictionary Keys

- **Dictionary values have no restrictions**. They can be any arbitrary Python objects, either standard objects or user-defined objects.

You can even use lists and tuples as values, a dictionary just doesn't care.

- However, same is not true for the keys.

Dictionaries (Allowable Key Types)

```
dict = {['Name']: 'Zara', 'Age': 7}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/9x/hx0451rj4n1fcxsm1m6ch5hx7p3m0h/T/ipykernel_10177/1843431795.py in <module>
----> 1 dict = {['Name']: 'Zara', 'Age': 7}

TypeError: unhashable type: 'list'
```

The key provided above is a mutable type. It is a list.

The interpreter will raise an error
Indicating that a list is an unhashable type

Why is that?

- Under the hood, dictionaries are powered by hash tables, array of slots:

```
daughterFavouriteFruits = {"Banana":"For porridge”,
        "Avocado":"For smoothies", "Strawberry": "To eat it raw"}
```

Hashing index	Value
7899950176585080344	For porridge
-1788313497634116637	For smoothies
4590764003693650766	To eat raw

How is it so fast?

- My daughter wasn't introduced to many fruits, so the number of slots in her list is 3, however a hash table must have empty slots.
- Let's consider a size of 8:

hash("Banana")%8 -> 0
hash("Avocado")%8 -> 3
hash("Straweberry")%8 -> 6

7899950176585080344 %8 -> 0
-1788313497634116637 % 8 -> 3
45907640036936507 66 % 8 -> 6

Hashing index	Value
0	For porridge
1	
2	
3	For smoothies
4	
5	
6	To eat raw
7	

Dictionaries (Using a list as the value element in a dictionary and the len method)

```
testScore = {'Scully':[56, 67, 34,],
             'Fitzgibbon':[78, 76, 54]}
print (testScore['Scully'])
print (len(testScore))
```

```
[56, 67, 34]
2
```

Dictionaries (Using a for loop)

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

for key in ages:
    print (ages[key])
```

```
23
19
78
45
```

Some Dictionary Methods

- `clear` method: deletes all the elements in a dictionary, leaving it empty
  - Format: *`dictionary`*`.clear()`
- `keys` method: returns all the dictionary's keys as a list
  - Format: *`dictionary`*`.keys()`
- `values` method: returns all the values in the dictionary as a list
  - Format: *`dictionary`*`.values()`
  - Use a `for` loop to iterate over the values

Dictionaries (Using the keys method)

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

print ("The following people are in the dictionary:")
print (ages.keys())
```

```
The following people are in the dictionary:
dict_keys(['Sue', 'Peter', 'Andrew', 'Karren'])
```

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Sets

Sets

- Set is an unordered collection of items.
  - All items must be **unique**. No two elements can have the same value.
  - Set is **unordered and immutable.**
  - Set can be used to perform mathematical operations!!

Creating a Set

- A set is create using the following notation:

```
my_set = {x,x}
```

- Or by using the built-in set() function:

```
my_set = set([x,x])
```

Sets (Creation)

- Creating a set

```
set1 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
set2 = set(['hello', 'there'])
print(set1)
print(set2)
print(type(set1), type(set2))
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{'hello', 'there'}
<class 'set'> <class 'set'>
```

- Iterating a set

```
for x in set2:
    print(x)
```

```
hello
there
```

- len function

```
print(len(set1))
```

```
9
```

- Clearing a set

```
set1.clear()
print(set1)
```

```
set()
```

Union of Sets

- To find the union of two sets:
  - Use the `union` method
    - Format: *set1*`.union(`*set2*`)` or `set1 | set2`
    - Returns a new set which contains the union of both sets

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])
set3 = set([4, 7, 8, 9])

set4 = set1 | set2 | set3
print (set4)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Intersection of Sets

- Intersection of two sets: a set that contains only the elements found in both sets
  - Use the intersection method
    - Format: set1.intersection(set2) or set1 & set2

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])
set3 = set([4, 7, 8, 9])

set4 = set1 & set2 & set3
print (set4)
```

```
{4}
```

Finding the Difference of Sets

- Difference of two sets: a set that contains the elements that appear in the first set but do not appear in the second set
- To find the difference of two sets:
  - Use the `difference` method
    - Format: *`set1`*`.difference(`*`set2`*`)`
  - Use the – operator
    - Format: *`set1 - set2`*

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])

difference = set1 - set2;
print (difference)
```

```
{1, 2, 3}
```

Finding Subsets and Supersets

- To determine whether set A is subset of set B
  - Use the `issubset` method
    - Format: *setA*`.issubset(`*setB*`)`
  - Use the <= operator
    - Format: *setA* <= *setB*

- To determine whether set A is superset of set B
  - Use the `issuperset` method
    - Format: *setA*`.issuperset(`*setB*`)`
  - Use the >= operator
    - Format: *setA* >= *setB*

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])

subset = set1 <= set2;
print (subset)
```

```
False
```

```
set1 = set([1, 2, 3, 4])
set2 = set([2, 4])

superset = set1 >= set2;
print (superset)
```

```
True
```

Finding the Symmetric Difference of Sets

- Symmetric difference of two sets: a set that contains the elements that are not shared by the two sets
- To find the symmetric difference of two sets:
  - Use the `symmetric_difference` method
    - Format: *set1*`.symmetric_difference(`*set2*`)`
  - Use the ^ operator
    - Format: *set1* ^ *set2*

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])

difference = set1 ^ set2;
print(difference)
```

```
{1, 2, 3, 5, 6}
```

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

That’s all folks!