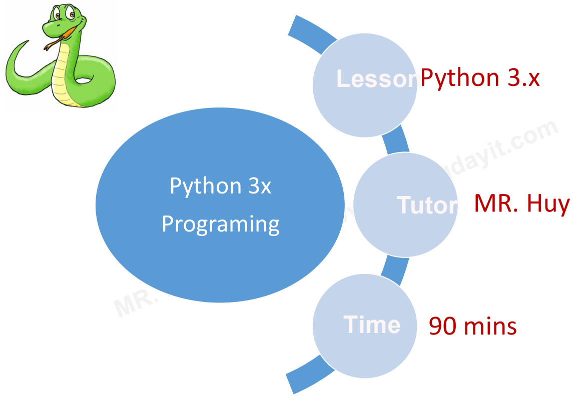


Python 3.x

MR. Huynh Nam

90 mins

Content

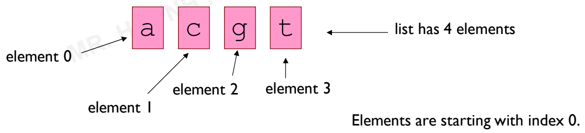
- Python Data Type

Data type - Python List

- A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.). Also, a list can even have another list as an item. This is called nested list.

```
nucleotides = ['a', 'c', 'g', 't']
print ("List of Nucleotides: ", nucleotides)
```

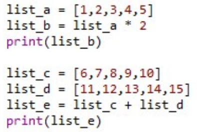
```
List of Nucleotides: ['a', 'c', 'g', 't']
```



Elements are starting with index 0.

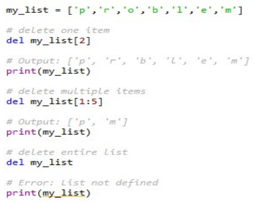
Data type - Python List

- Access item by index
- -1 is index which is considered the last element in list
- Operators: + (combine), * (repeat) → result is a list



```
list_a = [1,2,3,4,5]
list_b = list_a * 2
print(list_b)

list_c = [6,7,8,9,10]
list_d = [11,12,13,14,15]
list_e = list_c + list_d
print(list_e)
```

- Delete one or more items from a list using the keyword **del**. It can even delete the list entirely.



```
my_list = ['p','r','o','b','l','e','m']

# delete one item
del my_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)

# delete multiple items
del my_list[1:5]

# Output: ['p', 'm']
print(my_list)

# delete entire list
del my_list

# Error: List not defined
print(my_list)
```

Data type - Python List

- Methods: append, extend, insert, remove, pop, index, count, sort, reserve

```
x = ['a', 'c', 'g', 't']
i=2
print (x[0], x[i], x[-1])
```

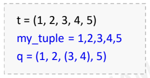
```
a g t
```

```
x = ['a', ‘c', ‘b', ‘d']
print ("x =",x)
x.sort()
print ("x =",x)
x.reverse()
print ("x =",x)
```

```
x = a c b d
x = a b c d
x = d c b a
```

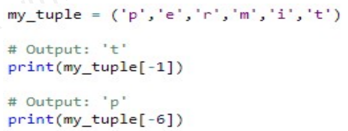
Data type - Python Tuple

- A tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.
- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).



```
t = (1, 2, 3, 4, 5)
my_tuple = 1,2,3,4,5
q = (1, 2, (3, 4), 5)
```

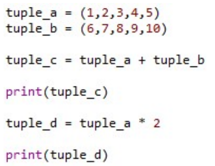
- Negative index: allows negative indexing for its sequences.



```
my_tuple = ('p','e','r','m','i','t')

# Output: 't'
print(my_tuple[-1])

# Output: 'p'
print(my_tuple[-6])
```

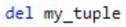
Data type - Python Tuple

- Concatenation (+): combine two tuples
- Repeat (*): repeat the elements in a tuple for a given number of times
- → result is new tuple



```
tuple_a = (1,2,3,4,5)
tuple_b = (6,7,8,9,10)

tuple_c = tuple_a + tuple_b

print(tuple_c)

tuple_d = tuple_a * 2

print(tuple_d)
```

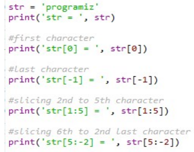
- Keyword “del”: deleting a tuple entirely



```
del my_tuple
```

Data type - Python String

- A string is a sequence of characters.
- Single / double quotation mark or : ‘, “
- Combining string: +, +=
- Access characters and slice: []



```
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

```
a = "pan"
b = "cake"
a = a + b
print (a)
c = “hello”
d = “world”
c += d
print(c)
```

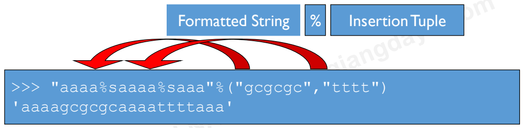
```
pancake
```

```
helloworld
```

- Get string length x: len(x)
- Strings are immutable. This means that **elements of a string** cannot be changed once it has been assigned. We can simply **reassign different strings** to the same name.
- Keyword “**del**”: deleting a string entirely

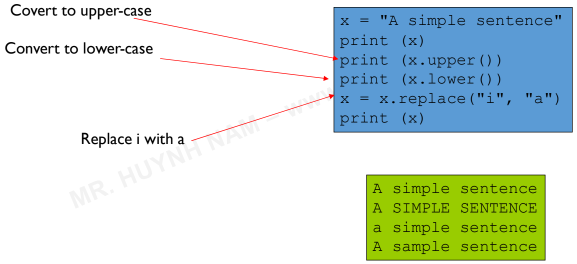
Data type - Python String

- Format string:
  - Some methods: upper, lower, split, replace …



```
>>> "aaaa%saaaa%saaa"%("gcgcgc","tttt")
'aaaagcgcgcaaaattttaaa'
```

Data type - Python String

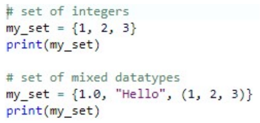


```
x = "A simple sentence"
print (x)
print (x.upper())
print (x.lower())
x = x.replace("i", "a")
print (x)
```

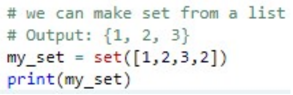
```
A simple sentence
A SIMPLE SENTENCE
a simple sentence
A sample sentence
```

Data type - Python Sets

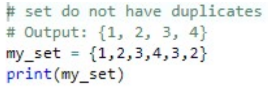
- A set is an **unordered** collection of items. Every element is unique (**no duplicates**) and must be immutable (which **cannot be changed**).
- However, the set itself is mutable. We can add or remove items from it.
- A set is created by placing all the items (elements) inside curly braces **{}**, separated by comma or by using the built-in function set().
- Cannot access or change an element of set using indexing or slicing.



```
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```



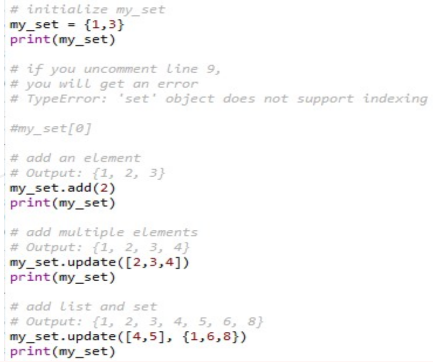
```
# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)
```



```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)
```

Data type - Python Sets

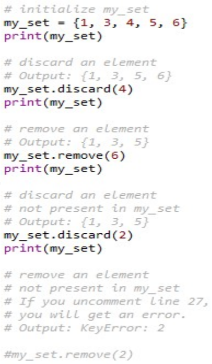
- Add single element using the **add()** method and multiple elements using the **update()** method.
- The **update()** method can take tuples, lists, strings or other sets as its argument.



```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

Data type - Python Sets

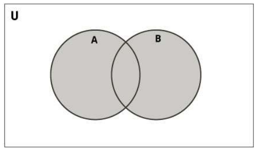
- Removed from set using methods, **discard()** and **remove()**.
- Using **discard()** if the item does not exist in the set, it remains unchanged. But **remove()** will raise an error in such condition.

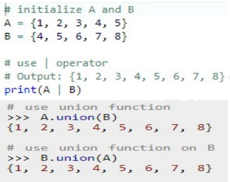


```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)

# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)

# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)

# remove an element
# not present in my_set
# If you uncomment line 27,
# you will get an error.
# Output: KeyError: 2

#my_set.remove(2)
```

Data type - Python Sets

- Set Union: |

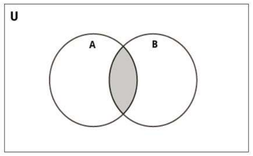


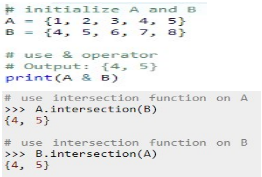


```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

```
# use union function
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

- Set Intersection: &



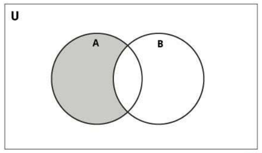


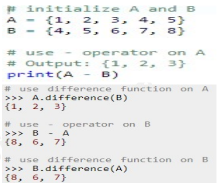
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}

# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

Data type - Python Sets

- Set Difference: -

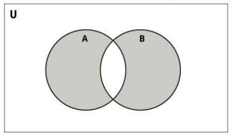


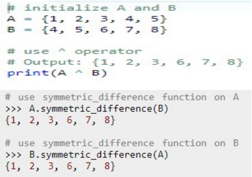


```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}

# use - operator on B
>>> B - A
{8, 6, 7}

# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

- Set Symmetric Difference: ^



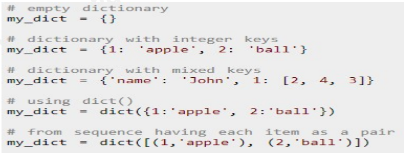


```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

# use symmetric_difference function on B
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

Data type - Python Dictionary

- Python dictionary is an unordered collection of items.
- A dictionary has a {**key: value**} pair
- Dictionaries are optimized to retrieve values when the key is known.
- Values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.
- Methods: keys, values, pop, items, has_key…



```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

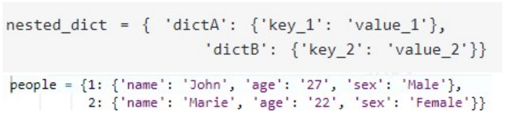
- Accessing valye by key with operator []

Read value: my_dict[‘name’] → **#Output: ‘John’**
Assign value: my_dict[‘name’] = “Nam“ → Output: ‘Nam’

Using operator [ ] to look up value

Data type – Python Nested Dictionary

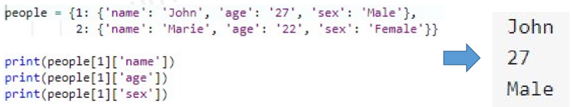
- A nested dictionary is a dictionary inside a dictionary. It's a collection of dictionaries into one single dictionary.



```
nested_dict = { 'dictA': {'key_1': 'value_1'},
                'dictB': {'key_2': 'value_2'}}
```

```
people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
          2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}
```

- Access the elements using the [] syntax

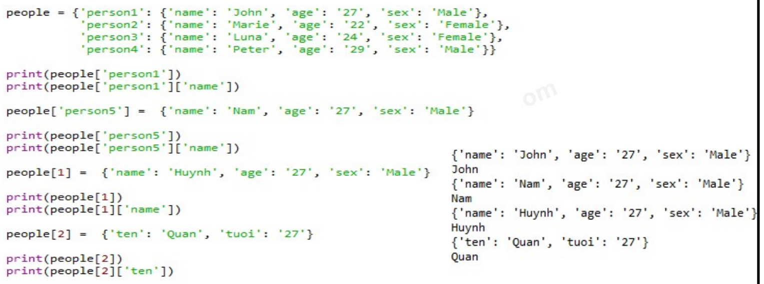


```
people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
          2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

print(people[1]['name'])
print(people[1]['age'])
print(people[1]['sex'])
```

```
John
27
Male
```

Data type – Python Nested Dictionary

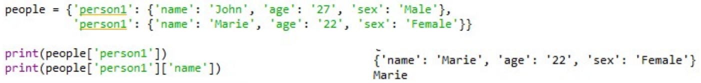
- Example



```
people = {'person1': {'name': 'John', 'age': '27', 'sex': 'Male'},
          'person2': {'name': 'Marie', 'age': '22', 'sex': 'Female'},
          'person3': {'name': 'Luna', 'age': '24', 'sex': 'Female'},
          'person4': {'name': 'Peter', 'age': '29', 'sex': 'Male'}}

print(people['person1'])
print(people['person1']['name'])

people['person5'] =  {'name': 'Nam', 'age': '27', 'sex': 'Male'}

print(people['person5'])
print(people['person5']['name'])

people[1] =  {'name': 'Huynh', 'age': '27', 'sex': 'Male'}

print(people[1])
print(people[1]['name'])

people[2] =  {'ten': 'Quan', 'tuoi': '27'}

print(people[2])
print(people[2]['ten'])
```

```
{'name': 'John', 'age': '27', 'sex': 'Male'}
John
{'name': 'Nam', 'age': '27', 'sex': 'Male'}
Nam
{'name': 'Huynh', 'age': '27', 'sex': 'Male'}
Huynh
{'ten': 'Quan', 'tuoi': '27'}
Quan
```

- What is result?



```
people = {'person1': {'name': 'John', 'age': '27', 'sex': 'Male'},
          'person1': {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

print(people['person1'])
print(people['person1']['name'])
```

```
{'name': 'Marie', 'age': '22', 'sex': 'Female'}
Marie
```

Review

- LIST: Access, Index, Slicing
  - Tuple
  - String
- SET
- DICTIONARY
- → Access

THANK YOU

Q & A