

Table of Contents

Objectives	3
1 Python Lists	3
1.1 Indexing & Slicing Examples	3
1.2 List Modification Examples.....	4
1.3 List functions	4
1.4 List Comprehensions	5
2 Python Tuples	6
2.1 Tuple Functions	7
3 Python Dictionaries.....	7
3.1 Dictionary Modification Examples.....	8
3.2 Dictionary Formatting Example	9
3.3 Dictionary Functions.....	9
4 Python Sets	10
4.1 Set Initialization Examples	10
4.2 Set Modification Examples	11
4.3 Set Operations	12

Objectives

After performing this lab, students shall be able to understand Python data structures which includes:

- ✓ Python lists
- ✓ Python tuples
- ✓ Python dictionaries
- ✓ Python sets

1 Python Lists

Everything in Python is treated as an object. Lists in Python represent ordered sequences of values. Lists are "mutable", meaning they can be modified "in place". You can access individual list elements with square brackets. Python uses *zero-based* indexing, so the first element has index 0.

Here are a few examples of how to create lists:

```
# List of integers primes
= [2, 3, 5, 7]

# We can put other types of things in lists
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', '
Uranus', 'Neptune']

# We can even make a list of lists hands
= [
    ['J', 'Q', 'K'],
    ['2', '2', '2'],
    ['6', 'A', 'K'], # (Comma after the last element is optional)
]

# A list can contain a mix of different types of variables:
my_favourite_things = [32, 'AI Lab, 100.25]
```

1.1 Indexing & Slicing Examples

Consider our list of planets created above:

```

planets[0] # 'Mercury'
planets[1] # 'Venus'
planets[-1] # 'Neptune'
planets[-2] # 'Uranus'

# List Slicing

# first three planets planets[0:3] #
['Mercury', 'Venus', 'Earth'] planets[:3] #
['Mercury', 'Venus', 'Earth']

# All the planets from index 3 onward
planets[3:] # ['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
# All the planets except the first and last
planets[1:-1] # ['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']

# The last 3 planets
planets[-3:] # ['Saturn', 'Uranus', 'Neptune']

```

1.2 List Modification Examples

Working with the same planets list:

```

# Rename Mars
planets[3] = 'Malacandra'
# ['Mercury', 'Venus', 'Earth', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

# Rename multiple list indexes planets[:3]
= ['Mur', 'Vee', 'Ur']
['Mur', 'Vee', 'Ur', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

```

1.3 List functions

Python has several useful functions for working with lists.

```

len(planets) # 8

# The planets sorted in alphabetical order sorted(planets)

```

```

# ['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']

primes = [2, 3, 5, 7]
sum(primes) # 17 max(primes)
# 7

# Let's add Pluto to the planets list planets.append('Pluto')

# Pop removes and returns the last element of the list planets.pop()
# 'Pluto'

# Remove an item from a list given its index instead of its value
a = [-1, 1, 66.25, 333, 333, 1234.5] del a[0] # [1, 66.25, 333, 333, 1234.5]

# Remove slices from the list del
a[2:4] # [1, 66.25, 1234.5]

planets.index('Earth') # 2

# Is Earth a planet?
"Earth" in planets # True

# Is Pluto a planet?
"Pluto" in planets # False (We removed it remember)

# Finally to find all the methods associated with Python list object
help(planets)

```

1.4 List Comprehensions

List comprehensions are one of Python's most unique features. List comprehensions combined with functions like min, max, and sum can lead to impressive one-line solutions for problems that would otherwise require several lines of code. The easiest way to understand them is probably to just look at a few examples:

```

# With list comprehension                                # [0, 1, 4, 9, 16, 25, 36, 49
squares = [n**2 for n in range(10)]
, 64, 81]

# Without list comprehension
squares = []
for n in range(10):
    squares.append(n**2)

# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# List comprehensions are great of filtering and transformations
short_planets = [planet for planet in planets if len(planet) < 6]
# ['Venus', 'Earth', 'Mars']

[
    planet.upper() + '!'
for planet in planets      if
len(planet) < 6
]
# ['VENUS!', 'EARTH!', 'MARS!']

# One line solution def
count_negatives(nums):
    # False + True + True + False + False equals to 2.
    # return len([num for num in nums if num < 0])
    return sum([num < 0 for num in nums])

count_negatives([5, -1, -2, 0, 3])

```

2 Python Tuples

Tuples are almost exactly the same as lists. They differ in just two ways.

1. The syntax for creating them uses parentheses instead of square brackets.
2. They cannot be modified (they are *immutable*).

Tuples are often used for functions that have multiple return values.

```

t = (1, 2, 3)
t = 1, 2, 3 # equivalent to above
t[0] = 100 # TypeError: 'tuple' object does not support item assignment

# Classic Python Swapping Trick
a = 1 b = 0
a, b = b, a # 0 1

```

2.1 Tuple Functions

There are only two tuple methods `count()` and `index()` that a tuple object can call.

```

thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5) x
= thistuple.count(5) # 2

thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5) x
= thistuple.index(8) # 3

```

3 Python Dictionaries

Dictionaries and lists share the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries differ from lists primarily in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys not by numerical index.

Duplicate keys are not allowed. A dictionary key must be of a type that is immutable. E.g. a key cannot be a list or a dict.

Here are a few examples to create dictionaries:

```

MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'
}
# Can also be defined as:
MLB_team = dict([
    ('Colorado', 'Rockies'),
    ('Boston', 'Red Sox'),
    ('Minnesota', 'Twins'),
    ('Milwaukee', 'Brewers'),
    ('Seattle', 'Mariners')
])
# Another way
tel = dict(sape=4139, guido=4127, jack=4098)

# dict comprehensions can be used to create dictionaries from arbitrary key and value expression
{x: x**2 for x in (2, 4, 6)}      # {2: 4, 4: 16, 6: 36}
# Building a dictionary incrementally - if you don't know all the key-value pairs in advance
person = {}
person['fname'] = 'Joe'
person['lname'] = 'Fonebone'
person['age'] = 51
person['spouse'] = 'Edna'
person['children'] = ['Ralph', 'Betty', 'Joey']
person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}
# {'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
#  'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

```

3.1 Dictionary Modification Examples

A few examples to access the dictionary elements, add new key value pairs, or update previous value:

```

# Retrieve a value
MLB_team['Minnesota']      # 'Twins'

# Add a new entry
MLB_team['Kansas City'] = 'Royals'

```

```
# Update an entry
MLB_team['Seattle'] = 'Seahawks'
```

3.2 Dictionary Formatting Example

The % operator works conveniently to substitute values from a dict into a string by name:

```
hash = {}
hash['word'] = 'garfield' hash['count']
= 42
s = 'I want %(count)d copies of %(word)s' % hash # %d for int, %s for
string
# 'I want 42 copies of garfield'
```

3.3 Dictionary Functions

The following is an overview of methods that apply to dictionaries:

```
# Let's use this dict for to demonstrate dictionary functions d
= {'a': 10, 'b': 20, 'c': 30}

# Clears a dictionary. d.clear()
# {}

# Returns the value for a key if it exists in the dictionary.
print(d.get('b')) # 20

# Removes a key from a dictionary, if it is present, and returns its v
alue.
d.pop('b') # 20

# Returns a list of key-value pairs in a dictionary.
list(d.items()) # [('a', 10), ('b', 20), ('c', 30)]
list(d.items())[1][0] # 'b'
```



```
list(d.items())[1][1]          # 20

# Returns a list of keys in a dictionary. list(d.keys())
# ['a', 'b', 'c']

# Returns a list of values in a dictionary.
list(d.values())               # [10, 20, 30]

# Removes the last key-value pair from a dictionary. d.popitem()
# ('c', 30)

# Merges a dictionary with another dictionary or with an iterable of k
ey-value pairs.
d2 = {'b': 200, 'd': 400}
d.update(d2) # {'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

For more details, visit [iterate dictionary](#) & [dictionary comprehensions](#)

4 Python Sets

Sets have following characteristics:

- Set in Python is a data structure equivalent to sets in mathematics.
- Sets are a mutable collection of distinct (unique) immutable values that are unordered.
- Any immutable data type can be an element of a set: a number, a string, a tuple.
- Mutable (changeable) data types cannot be elements of the set.
- In particular, list cannot be an element of a set (but tuple can), and another set cannot be an element of a set.
- You can perform standard operations on sets (union, intersection, difference).

4.1 Set Initialization Examples

You can initialize a set in the following ways:

```

# Initialize empty set
emptySet = set()

# Pass a list to set() to initialize it
dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau',
'SAS']) dataEngineer = set(['Python', 'Java', 'Scala', 'Git',
'SQL', 'Hadoop'])

# Direct initialization using curly braces
dataScientist = {'Python', 'R', 'SQL', 'Git', 'Tableau',
'SAS'} dataEngineer = {'Python', 'Java', 'Scala', 'Git',
'SQL', 'Hadoop'}

# Curly braces can only be used to initialize a set containing
values emptyDict= {}

```

4.2 Set Modification Examples

Let's consider the following set for our add/remove examples:

```

# Initialize set with values
graphicDesigner = {'InDesign', 'Photoshop', 'Acrobat',
'Premiere', 'Bridge'}

# Add a new immutable element to the set
graphicDesigner.add('Illustrator')

# TypeError: unhasable type 'list'
graphicDesigner.add(['Powerpoint', 'Blender'])

# Remove an element from the set
graphicDesigner.remove('Illustrator')

# Another way to remove an element. What is the difference?
graphicDesigner.discard('Premiere')

# Remove and return an arbitrary value from a set
graphicDesigner.pop()

# Remove all values from the set
graphicDesigner.clear()

```

4.3 Set Operations

Python sets have methods that allow you to perform these mathematical operations like union, intersection, difference, and symmetric difference. Let's initialize two sets to work on our examples:

```
# Initialize sets
dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau',
'SAS']) dataEngineer = set(['Python', 'Java', 'Scala', 'Git',
'SQL', 'Hadoop'])

# set built-in function union
dataScientist.union(dataEngineer)

# Equivalent Result
dataScientist | dataEngineer

# Intersection operation
dataScientist.intersection(dataEngineer)

# Equivalent Result
dataScientist & dataEngineer

# These sets have elements in common so isdisjoint would return
False
dataScientist.isdisjoint(dataEngineer)

# Difference Operation
dataScientist.difference(dataEngineer)

# Equivalent Result
dataScientist - dataEngineer

# Symmetric Difference Operation
dataScientist.symmetric_difference(dataEngineer)

# Equivalent Result dataScientist ^ dataEngineer
```

EXERCISE QUESTIONS:

Ex:1.1

Indexing & Slicing

Create a list: `fruits = ['apple', 'banana', 'cherry', 'date', 'fig']`

Print:

- First element
- Last element
- First three elements
- Last two elements

Ex:1.2

List Modification

Using the same fruits list:

- Replace the second element with 'blueberry'
- Delete the last element
- Add a new element 'grape'

Ex:1.3

List Functions

Create a numeric list: `[10, 5, 7, 3, 9]`

Find:

- Sum of the list
- Maximum number
- Minimum number
- Sort the list in ascending order

Ex:1.4

List Comprehension

Create a list of **squares of even numbers** from 1 to 10 using a one-liner list comprehension

Create a list containing only the fruits whose names have **less than 6 characters**

Ex:1.5

Tuple Creation & Access

Create a tuple: `colors = ('red', 'green', 'blue', 'yellow')`

Access:

- First element
- Last element
- Slice from the second to the third element

Try modifying: `colors[1] = 'black'` → Observe the error

Ex:1.6

Tuple Functions

Create a tuple: (1, 2, 2, 3, 4, 2, 5)

Find:

How many times 2 appears

The index of the first 3

Ex:1.7

Dictionary Access & Modification

Create a dictionary: `student = {'name':'Ali', 'age':20, 'grade':'A'}`

Print:

Student's name

Add a new key 'city' = 'Lahore'

Update 'grade' = 'A+'

Delete the 'age' key

Ex:1.8

Dictionary Functions & Formatting

Create a dictionary: `scores = {'Math':90, 'Physics':85, 'Chemistry':92}`

Print:

List of all keys

List of all values

Total score (sum of all values)

Format a string using the dictionary: "Ali scored 90 in Math" dynamically using scores

Ex:1.9

Set Initialization & Modification

Create a set: `tools = {'Photoshop', 'Illustrator', 'Acrobat'}`

Add 'Premiere'

Remove 'Acrobat'

Try adding a list ['InDesign','Bridge'] → Observe the error

Clear the set completely

Ex:1.10

Set Operations

Create two sets:

`DS_skills = {'Python', 'R', 'SQL', 'Git'}`

`DE_skills = {'Python', 'Java', 'SQL', 'Hadoop'}`

Find:

Union of both sets

Intersection of both sets

Difference: `DS_skills - DE_skills`

Symmetric difference of both sets