MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT9133 Semester 1 2019
Programming Foundations in Python

Week 3:
Introduction to Data Structures, Collective Data Types

Cunyang Chen
Gavin Kroeger

GROUP
OF EIGHT
AUSTRALIA

Assignment

- Released on Moodle
  - PDF descriptions and 3 template source-code files
  - Ask for any explanation or clarification in the lab or forum

- Submission
  - Due: September 8th, 2019, 11:55pm.
  - Submission point to be created this week.
  - No plagiarism.
  - Please begin your assignment asap as assignment from other units will also come soon.

A revisit to “==”

- “123” == 123: False
  - The == operator returns True if there is an exact match, otherwise False will be returned

- “123” > 123: TypeError
  - To see if one value is greater than the other, they must be comparable i.e., of the similar data type.

Iteration Constructs

- `for` loop:
  - Similar to `while` loop; except that the governing condition (logical expression) does not need to be defined
  - Useful for iterating or traversing through a collection of items (e.g. Lists)

```
num_list = [1, 2, 3, 4, 5]
product = 1
for item in num_list:
    product *= item
print(product)
```

- Note: This special `for` loop structure with the `in` operator can be used on any *iterable* collective data type (e.g. Dictionary).

Module 2 Synopsis

- Module 2 is aimed to provide you with:
  - Concepts of data structure and data type
  - Collective data types in Python:
    - Strings
    - Sequences: List, Tuple, Set and Dictionary
    - Built-in methods

Module 2 Learning Objectives

- Upon completing this module, you should be able to:
  - Recognise key differences between data structure and data type
  - Deploy a suitable Python built-in data type for the representation of a particular form of data
  - Control the flow of program execution with selective and iterative structures

Data Types and Data Structures

Primitive Data Types

- Two types of (built-in) data types in Python:
  - Atomic
  - Collective

- Atomic types:
  - *Indivisible* which represents only a single data value
  - E.g. Integer (`int`), floating-point (`float`), Boolean (`bool`)

- Collective types:
  - Collections of multiple data values
  - E.g. String (`str`), List (`list`), Tuple (`tuple`), Set (`set`), Dictionary (`dict`)

complex data types

Data Structures

- Data structures:
  - Define how the data values represented by a particular data type are physically stored and organized in the memory
  - Specific how individual data values within a collection can be accessed and manipulated

- Difference between data type and data structure:
  - Data structure is a general term of theoretical computer science
  - Data type is more about the implementation of data structure in certain programming language

Data Structures

- Atomic data structures types:
  - Primitive e.g., Boolean, Floating-point, Integer
  - Non-primitive e.g., array, list
    - Linear list e.g., stack, queue
    - Non-linear list e.g., graph, tree, hashmap

- Collective data structure in Python:
  - **`str`** :`a = “hello”`
  - **`list:`**`a = [3, 1, 5]`
  - **`tuple`**: `a = (1, 7)`
  - **`set:`** `a = {1, 5, 3}`
  - **`dictionary`**: `a = {1:”a”, 2:”b”}`

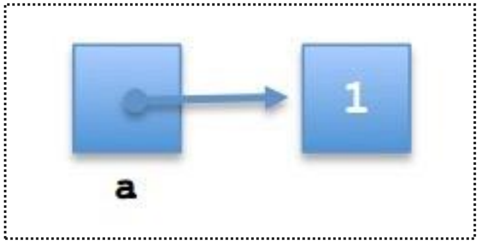
MONASH University

Python Collective Data Types

Memory and Data Storage

- Memory:
  - A collection of *contiguous* blocks of data storage
- Information stored in memory blocks are either:
  - An object in Python (with its data values)
  - An reference to the memory address (location) of another memory block

```
a = 1
```

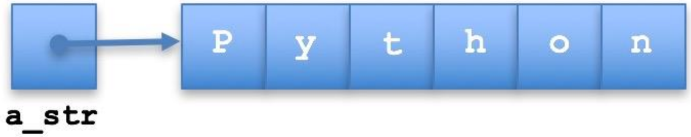


Array-based Structure

- Array-based structure:
  - Data items of a collection are organised and stored sequentially in a contiguous block of memory
  - Individual data items are located at the *adjacent* memory blocks
  - Each data item can be *randomly* accessed using the concept of “indexing”

Example: Array-based Structure

- String data type:
  - Assuming implemented using the array-based structure
  - E.g. `a_str = “Python”`



- Accessing individual items of an array-based structure:
  - `[]` is used to indicate the position (index) of a specific item within the collection
  - First character: `a_str[0]`
  - Last character: `a_str[len(a_str)-1]`

String Data Type

- String (Python type **`str`**):
  - A data type for textual representation
  - A list or collection of characters “strung” together to form strings (or sentences)

- Concatenation of strings:
  - Attach individual strings to one another to form larger strings
  - With the use of *overloaded* ‘**+**’ operator

str.join()

```
>>> first_name = “Chunyang”
>>> last_name = “Chen”
>>> full_name = first_name + “ ” + last_name
>>> print(full_name)
>>> ‘Chunyang Chen’
```

(More on) String Data Type

- Accessing characters of Python strings:
  - Use of a pair of “**[]**”
  - Python indices begin at **0** (not **1**)
  - https://en.wikipedia.org/wiki/Zero-based_numbering

- Slicing in Python strings:
  - Allow large chunks of a string (substrings) to be accessed
  - Syntax: **a_str[start_index:end_index]**
  - Sliced substrings are new String objects (without modifying the original string)

```
>>> message = “Welcome to FIT9133”
>>> sub_message = message[0:7]
>>> print(sub_message)
>>> ‘Welcome’
```

(More on) String Data Type

- Built-in String methods:

```
>>> message = “Welcome to FIT9133”
>>> message.split()
>>> [‘Welcome’, ‘to’, ‘FIT9133’]
>>> message = “  Welcome to FIT9133  ”
>>> message = message.strip()
>>> print(message)
>>> ‘Welcome to FIT9133’
>>> message.replace(‘o’, ‘0’)
>>> ‘Welc0me t0 FIT9133’
>>> message.isalpha()
>>> False
>>> message.isdigit()
>>> False
```

- Python 3 documentation for String type:
  - https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

(More on) String Data Type

- Commonly used string methods
  - str.capitalize()
  - str.count()
  - str.endswith()
  - str.find()
  - str.index()
  - str.isdigit()
  - str.join()
  - str.lower()
  - str.replace()
  - str.split()
  - str.strip()
  - ……..
  - https://docs.python.org/3/library/stdtypes.html#string-methods

Review Question 1

What would be the output for the given program?

```
simple_string = “This is a sentence”
simple_string[5] = ‘I’
print(simple_string)
```

Strings are immutable

A. This is a sentence
B. This Is a sentence
C. Some error occurred
D. Not sure

Review Question 2

What is the result of executing the given program?

```
simple_string = “This is a sentence”
simple_string.split(“ ”, 2)
```

A. [‘This’, ‘is a sentence’]

B. [‘This’, ‘is’, ‘a sentence’]

C. [‘This’] [‘is’, ‘a’, ’sentence’]

D. Some error occurred

E. Not sure

https://docs.python.org/3/library/stdtypes.html#str.split

Sequences

- Sequences in Python:
  - A collection of data items that enables read and/or manipulate the data items stored within the collection
  - E.g. List (`list`), Tuple (`tuple`), Set (`set`), Dictionary (`dict`)

- Common functionality of sequence-based types:
  - `in` statement
  - Concatenation
  - Indexing
  - Slicing
  - Basic analysis

Sequences: Common Functionality

- **in** statement:
  - Check for the existence of a specific item (element) in a sequence

```
if x in seq:
    print(“x exists”)
if x not in seq:
    print(“x doesn’t exist”)
```

- Concatenation:
  - Combine sequences of the same type together using the overloaded ‘**+**’ operator
  - Create a new sequence which has to be assigned to a new variable
  - E.g: **`new_seq = seq_one + seq_two`**

Sequences: Common Functionality

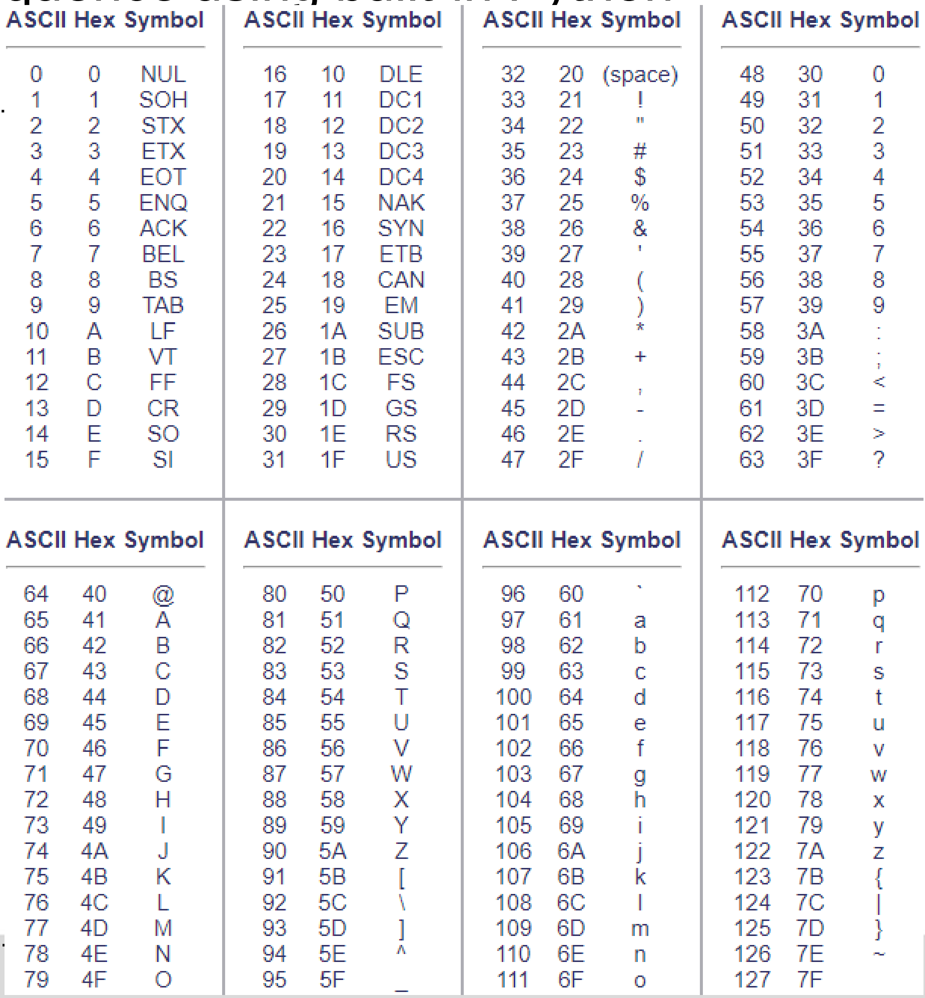
- Indexing:
  - Items (elements) in a sequence are accessed by using indexing with the first index as `0`
  - For *mutable* sequences, indexing can be used to change the value of the item (element) at a given location
  - Syntax: `a_seq[index]` or `a_seq[index] = new_value`

- Slicing:
  - Extract subsections of a sequence from the start index until the one before the end index
  - Syntax: `a_seq[start_index:end_index]` or `a_seq[start_index:end_index:step]`

Sequences: Common Functionality

- Basic analysis:
  - Investigate the contents of a sequence using built-in Python methods

```
>>> message = “Welcome to FIT9133”
>>> len(message)
>>> 18
>>> message.index(‘o’)
>>> 4
>>> message.count(‘o’)
>>> 2
>>> min(message)
>>> ‘ ’
>>> max(message)
>>> ‘t’
>>> message[0:7]
>>> ‘Welcome’
>>> message[0:7:2]
>>> ‘Wloe’
>>> print(message)
>>> ‘Welcome to FIT9133’
```



ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Collective Data Types:
List, Tuple

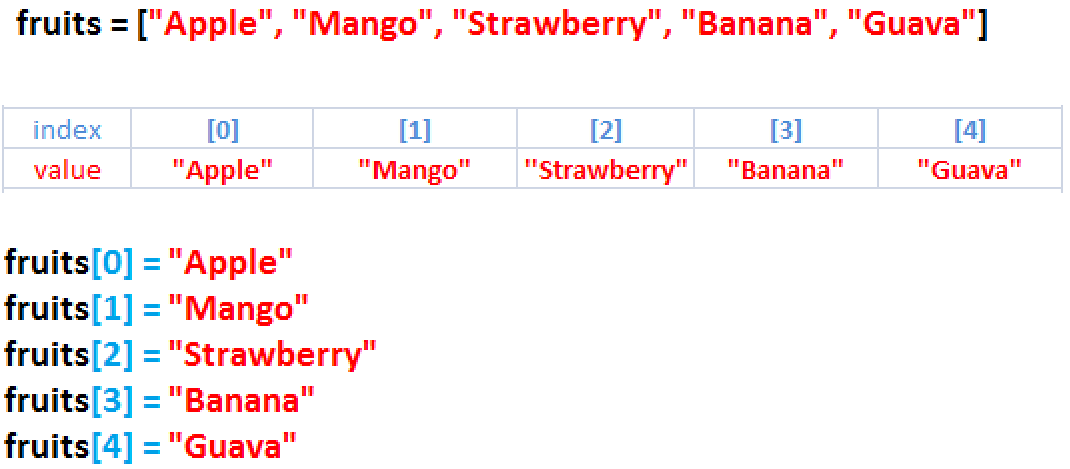
List Data Type

- List (Python type `list`):
  - A data type for a collection items which are generally related
  - A list can hold data objects of any data type
  - Mutable (changeable), ordered sequence

- Creating a list:
  - An empty list: `a_list = []` or `a_list = list()`
  - A list with a number of different items:
    `a_list = [1, ‘two’, 3.0, ‘4’]`
  - A list with a number of same items: `a_list = [1] * 6`

List Data Type

- The index of the list:
  - Index starts from 0



```
fruits = ["Apple", "Mango", "Strawberry", "Banana", "Guava"]
```

index	[0]	[1]	[2]	[3]	[4]
value	"Apple"	"Mango"	"Strawberry"	"Banana"	"Guava"

```
fruits[0] = "Apple"
fruits[1] = "Mango"
fruits[2] = "Strawberry"
fruits[3] = "Banana"
fruits[4] = "Guava"
```

- Adding new items to a list:
  - Append at the end of the list: `a_list.append(new_item)`
  - Insert at a specific position: `a_list.insert(index, new_item)`

(More on) List Data Type

- Removing items from a list:
  - `pop()`: accept one *optional* argument indicating the position of the item to be removed; otherwise remove the last item if no argument provided
  - `remove()`: accept one argument indicating the item value to be removed (not the position of the item); and delete the first instance of that item

- Manipulating items in a list:
  - `sort()`: sort the items of a list in place; by default in ascending order
  - `reverse()`: reverse the items of a list in place

https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

(More on) List Data Type

- Examples on manipulating a Python list:

```
>>> num_list = [3, 4, 2, 6]
>>> num_list.append(2)
>>> num_list.insert(2, 5)
>>> print(num_list)
>>> [3, 4, 5, 2, 6, 2]
>>> num_list.pop()
>>> 2
>>> num_list.pop(2)
>>> 5
>>> print(num_list)
>>> [3, 4, 2, 6]
>>> num_list.sort()
>>> print(num_list)
>>> [2, 3, 4, 6]
>>> num_list.reverse()
>>> print(num_list)
>>> [6, 4, 3, 2]
```

Review Question 3

What would be the output for the given program?

```
first_list = [‘This’, ‘is’, ‘a’, ‘sentence’]
second_list = [‘yes’, ‘or’, ‘no’]
first_list.extend(second_list)
print(first_list)
```

A. [‘This’, ‘is’, ‘a’, ‘sentence’]

B. [‘This’, ‘is’, ‘a’, ‘sentence’, ‘yes’, ‘or’, ‘no’]

C. [‘yes’, ‘or’, ‘no’, ‘This’, ‘is’, ‘a’, ‘sentence’]

D. Some error occurred

E. Not sure

Review Question 4

What is the result of executing the given program?

```
num_list = [3, -4, 2, 5, 1]
num_list.sort(reverse=True)
```

A. [3, -4, 2, 5, 1]

B. [-4, 1, 2, 3, 5]

C. [5, 3, 2, 1, -4]

D. Some error occurred

E. Not sure

Iteration Constructs

- `for` loop with `range()`:
  - For looping a certain number of times using an index; without having to increment the index
  - Syntax: `range(start_index, end_index, step)`
  - `E.g., list(range(5))= [0, 1, 2, 3, 4]`
  - `list(range(1, -3, -1)) = [1, 0, -1, -2]`

```
num_list = [1, 2, 3, 4, 5]
product = 1
for i in range(len(num_list)):
    product *= num_list[i]
print(product)
```

Iteration Constructs

- List comprehension:
  - For constructing a new list from an existing list
  - Items of the new list are built from applying some form of operations on the items of the existing list

```
num_list = [1, 2, 3, 4, 5]
odd_list = [each for each in num_list if each % 2 != 0]
print(odd_list)
```

“Pythonic” style

```
num_list = [1, 2, 3, 4, 5]
odd_list = []
for each in num_list:
    if each % 2 != 0:
        odd_list.append(each)
print(odd_list)
```

Review Question 5

What is the result of executing the given program?

```
mixed_list = [1, "2", 3.0, 4, “5”]
new_list = [each for each in mixed_list if type(each) is int]
print(new_list)
```

A. [1, 3.0, 4]

B. [1, 4]

C. [1, 2, 3.0, 4, 5]

D. Not sure

Review Question 6

What would be the output for the given program?

```
simple_list = [‘This’, ‘is’, ‘a’, ‘sentence’]
simple_list[2] = ‘A’
print(“ ”.join(simple_list))
```

A. This is a sentence
B. This is A sentence
C. ThisisAsentence
D. Some error occurred
E. Not sure

Tuple Data Type

- Tuple (Python type **`tuple`**):
  - A data type for “chunking” related information that belongs together (as a record)

- Creating a tuple:
  - An empty tuple: **`a_tuple = ()`**
  - A tuple with a number of items: **`a_tuple = (0,1)`**
  - Build a tuple from a list: **`a_tuple = tuple([0,1])`**

- Accessing individual items in a tuple:
  - Indexing: **`a_tuple[0]`**
  - Assigning to new variables: **`x, y = a_tuple`**
  - Note: Tuple does not support *item assignment,* immutable.

(More on) Tuple Data Type

- Examples on manipulating a Python tuple:

```
>>> xy_coord = (0, 1)
>>> print(xy_coord)
>>> (0, 1)
>>> print(xy_coord[0])
>>> 0
>>> print(xy_coord[1])
>>> 1
>>> x_coord, y_coord = xy_coord
>>> print(x_coord)
>>> 0
>>> print(y_coord)
>>> 1
>>> xy_coord[0] = 2
>>> TypeError
>>> xy_list = [(0,0),(0,1)]
>>> xy_tuples = tuple(xy_list)
>>> type(xy_tuples)
>>> <class 'tuple'>
```

Tuples are immutable

Difference between List and Tuple

- Tuples have structure, lists have order:
  - Tuples usually contain an heterogeneous sequence
  - E.g, `coordinates_tuple[0] = (1, 2) #(x, y)`
- Tuples are immutable, while lists are mutable:
  - Wrong: `a_tuple[0] = 1`
  - Yes: `a_list[0] = 1`
- Storing the same data, tuple owns smaller size:

```
a_tuple = tuple(range(1000))
b_list = list(range(1000))
a_tuple.__sizeof__()    #8024
b_list.__sizeof__()     #9088
```

Week 3 Summary

- So far, we have discussed:
  - Data structures vs data types
  - Collective data types (String, List, Tuple)

- Next week:
  - Other collective data types (Set, Dictionary)
  - Standard input and output in Python
  - Comments and documentations along with the source code

Reminder: Please get started with the first assessment.