CSC115

PYTHON PROGRAMMING

UNIVERSITY OF MIAMI

COLLEGE OF ARTS AND SCIENCES
COMPUTER SCIENCE

Chapter 7 & 8: Strings, Lists and Dictionaries

Instructor: Dr. Hien Nguyen

Topics

7.1	String slicing
7.2	Advanced string formatting
7.3	String methods
7.4	Splitting and joining strings
8.1	Lists
8.2	List methods
8.3	Iterating over a list
8.5	List nesting
8.6	List slicing
8.8	List comprehensions
8.9	Sorting lists
8.12	Dictionary
8.13	Dictionary methods
8.14	Iterating over a dictionary

7.1 String slicing

- index: An index is an integer matching to a specific position in a string's sequence of characters.
- Slice notation: Slice notation has the form **my_str[start:end]**

String slicing.

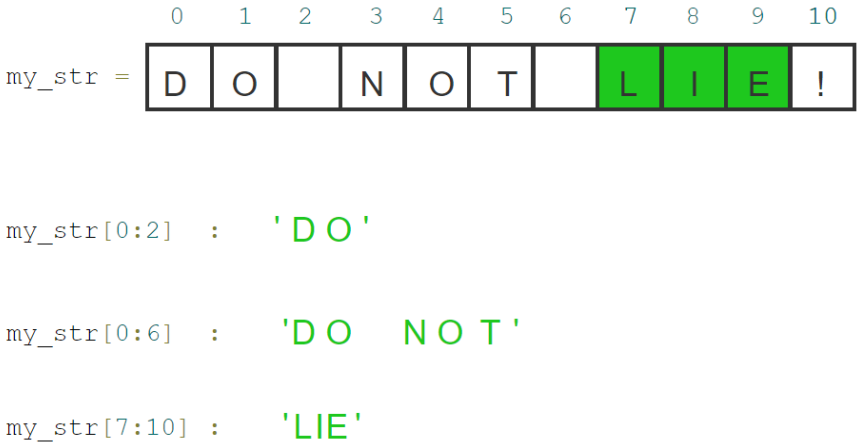
```
url = 'http://en.wikipedia.org/wiki/Turing'
domain = url[7:23]  # Read 'en.wikipedia.org' from url
print(domain)
```

```
en.wikipedia.org
```

7.1 String slicing

PARTICIPATION ACTIVITY | 7.1.1: Slicing.

Start ☐ 2x speed



Captions

1. my_str[0:2] returns a substring of my_str starting at index 0 up to, but not including, index 2.
2. my_str[0:6] returns a substring of my_str starting at index 0 up to, but not including, index 6.
3. my_str[7:10] returns a substring of my_str starting at index 7 up to, but not including, index 10.

7.2 Advanced string formatting

- Field width: A format specification may include a field width that defines the minimum number of characters that must be inserted into the string.

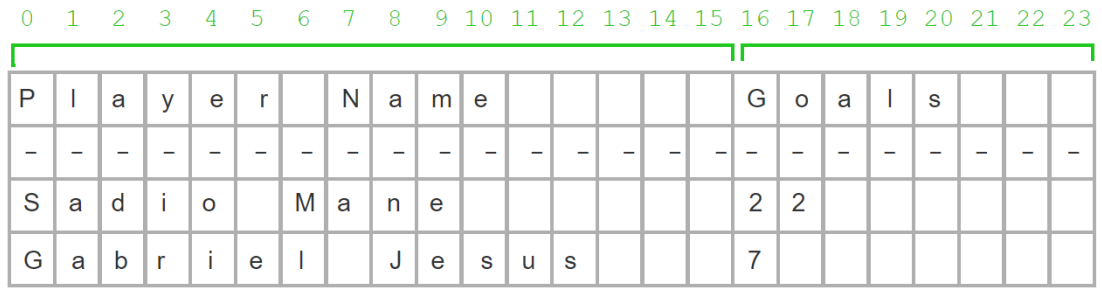
A formatted table of soccer statistics.

```
Player Name      Goals    Games Played     Goals Per Game
-------------------------------------------------------
Sadio Mane         22          36               0.61
Mohamed Salah      22          38               0.58
Sergio Aguero      21          33               0.64
Jamie Vardy        18          34               0.53
Gabriel Jesus       7          29               0.24
```

7.2 Advanced string formatting

```
print(f'{"Player Name":16}{"Goals":8}')
print('-' * 24)

print(f'{"Sadio Mane":16}{"22":8}')
print(f'{"Gabriel Jesus":16}{"7":8}')
```



Captions

1. 'Player Name' is inserted into the leftmost part of the first 16-character wide field. 'Goals' is inserted into the leftmost part of the second 8-character wide field.
2. The inserted values align themselves automatically according to the field width.

7.2 Advanced string formatting

- Alignment character: A format specification can include an alignment character that determines how a value should be aligned within the width of the field.

Consider the following code that prints a table, and how changing the alignment impacts the column organization.

```
names = ['Sadio Mane', 'Gabriel Jesus']
goals = [22, 7]

print(<f-string 1>)        #Replaced in table below
print('-' * 24)
for i in range(2):
    print(<f-string 2>)    #Replaced in table below
```

Alignment type	<f-string 1> <f-string 2>	Output

Left-aligned	`f'{"Player Name":<16}{"Goals":<8}'`
	`f'{names[i]:<16}{goals[i]:<8}'`
		`Player Name Goals`
		`------------------------`
		`Sadio Mane 22`
		`Gabriel Jesus 7`

Right-aligned	`f'{"Player Name":>16}{"Goals":>8}'`
	`f'{names[i]:>16}{goals[i]:>8}'`
		`Player Name Goals`
		`------------------------`
		`Sadio Mane 22`
		`Gabriel Jesus 7`

Centered	`f'{"Player Name":^16}{"Goals":^8}'`
	`f'{names[i]:^16}{goals[i]:^8}'`
		`Player Name Goals`
		`------------------------`
		`Sadio Mane 22`
		`Gabriel Jesus 7`

7.2 Advanced string formatting

- Fill character: The fill character is used to pad a replacement field when the string being inserted is smaller than the field width.

Using fill characters to pad tables.

Format specification	Value of score	Output
`{score:}`	9	`9`
`{score:4}`	9	`   9`
`{score:0>4}`	9	`0009`
`{score:0>4}`	18	`0018`
`{score:0^4}`	18	`0180`

7.2 Advanced string formatting

- Precision: The optional precision component of a format specification indicates how many digits to the right of the decimal should be included in the output of floating types.

```
import math
real_pi = math.pi  # math library provides close approximation of pi
approximate_pi = 22.0 / 7.0  # Approximately correct pi to within 2 decimal places

print(f'pi is {real_pi}')
print(f'22/7 is {approximate_pi}')
print(f'22/7 looks better like {approximate_pi:.2f}')
```

```
pi is 3.141592653589793
22/7 is 3.142857142857143
22/7 looks better like 3.14
```

7.3 String methods

- replace (old, new): Replace(old, new) -- Returns a copy of the string with all occurrences of the substring old replaced by the string new.
- replace (old, new, count) : Replace(old, new, count) -- Same as above, except only replaces the first count occurrences of old.

find(x)

Find(x) -- Returns the index of the first occurrence of item x in the string, else returns -1.

find(x, start)

Find(x, start) -- Same as find(x), but begins the search at index start.

find(x, start, end)

Find(x, start, end) -- Same as find(x, start), but stops the search at index end - 1.

rfind(x)

Rfind(x) -- Same as find(x) but searches the string in reverse, returning the last occurrence in the string.

count(x)

Count(x) -- Returns the number of times x occurs in the string.

7.4 Splitting and joining strings

split()

The string method split() splits a string into a list of tokens.

token

Each token is a substring that forms a part of a larger string.

separator

A separator is a character or sequence of characters that indicates where to split the string into tokens.

Figure 7.4.1: String split example.

```
url = input('Enter URL:\n')

tokens = url.split('/')  # Uses '/' separator
print(tokens)
```

```
Enter URL: http://en.wikipedia.org/wiki/Lucille_ball
['http:', '', 'en.wikipedia.org', 'wiki', 'Lucille_ball']
...
Enter URL: en.wikipedia.org/wiki/ethernet/
['en.wikipedia.org', 'wiki', 'ethernet', '']
```

7.4 Splitting and joining strings

join()

The join() string method performs the inverse operation of split() by joining a list of strings together to create a single string.

String join() method.

```
web_path = [ 'www.website.com', 'profile', 'settings' ]
separator = '/'
url = separator.join(web_path)
```

```
url = 'www.website.com/profile/settings'
```

Chapter 8

Lists and Dictionaries

8.1 Lists

- list: The list object type is one of the most important and often used types in a Python program.
- container: A list is a container, which is an object that groups related objects together.
- list(): The list() function accepts a single iterable object argument, such as a string, list, or tuple, and returns a new list object.
- index: an index is a zero-based integer matching to a specific position in the list's sequence of elements.

8.1 Lists

Table 8.1.1: Some common list operations.

Operation	Description	Example code	Example output
my_list = [1, 2, 3]	Creates a list.	`my_list = [1, 2, 3]` `print(my_list)`	`[1, 2, 3]`
list(iter)	Creates a list.	`my_list = list('123')` `print(my_list)`	`['1', '2', '3']`
my_list[index]	Get an element from a list.	`my_list = [1, 2, 3]` `print(my_list[1])`	`2`
my_list[start:end]	Get a *new* list containing some of another list's elements.	`my_list = [1, 2, 3]` `print(my_list[1:3])`	`[2, 3]`
my_list1 + my_list2	Get a *new* list with elements of my_list2 added to end of my_list1.	`my_list = [1, 2] + [3]` `print(my_list)`	`[1, 2, 3]`

my_list[i] = x	Change the value of the ith element in-place.	`my_list = [1, 2, 3]`
		`my_list[2] = 9`
		`print(my_list)`
			`[1, 2, 9]`

my_list[len(my_list):] = [x]	Add the elements in [x] to the end of my_list. The append(x) method (explained in another section) may be preferred for clarity.	`my_list = [1, 2, 3]`
		`my_list[len(my_list):] = [9]`
		`print(my_list)`
			`[1, 2, 3, 9]`

del my_list[i]	Delete an element from a list.	`my_list = [1, 2, 3]`
		`del my_list[1]`
		`print(my_list)`
			`[1, 3]`

8.2 List methods

List method	Description	Code example	Final my_list value
Adding elements			
list.append(x)	Add an item to the end of list.	`my_list = [5, 8]` `my_list.append(16)`	`[5, 8, 16]`
list.extend([x])	Add all items in [x] to list.	`my_list = [5, 8]` `my_list.extend([4, 12])`	`[5, 8, 4, 12]`
list.insert(i, x)	Insert x into list *before* position i.	`my_list = [5, 8]` `my_list.insert(1, 1.7)`	`[5, 1.7, 8]`
Removing elements			
list.remove(x)	Remove first item from list with value x.	`my_list = [5, 8, 14]` `my_list.remove(8)`	`[5, 14]`
list.pop()	Remove and return last item in list.	`my_list = [5, 8, 14]` `val = my_list.pop()`	`[5, 8]` val is 14
list.pop(i)	Remove and return item at position i in list.	`my_list = [5, 8, 14]` `val = my_list.pop(0)`	`[8, 14]` val is 5

8.2 List methods

list.sort()	Sort the items of list in-place.	`my_list = [14, 5, 8]` `my_list.sort()`	`[5, 8, 14]`
list.reverse()	Reverse the elements of list in-place.	`my_list = [14, 5, 8]` `my_list.reverse()`	`[8, 5, 14]`
list.index(x)	Return index of first item in list with value x.	`my_list = [5, 8, 14]` `print(my_list.index(14))`	Prints "2"
list.count(x)	Count the number of times value x is in list.	`my_list = [5, 8, 5, 5, 14]` `print(my_list.count(5))`	Prints "3"

8.3 Iterating over a list

Figure 8.3.2: Iterating through a list example: Finding the maximum even number.

```
user_input = input('Enter numbers:')

tokens = user_input.split()  # Split into separate strings

# Convert strings to integers
nums = []
for token in tokens:
    nums.append(int(token))

# Print each position and number
print()  # Print a single newline
for index in range(len(nums)):
    value = nums[index]

    print(f'{index}: {value}')

# Determine maximum even number
max_num = None
for num in nums:
    if (max_num == None) and (num % 2 == 0):
        # First even number found
        max_num = num
    elif (max_num != None) and (num > max_num ) and (num % 2 == 0):
        # Larger even number found
        max_num = num

print('Max even #:', max_num)
```

```
Enter numbers:3 5 23 -1 456 1 6 83
0: 3
1: 5
2: 23
3: -1
4: 456
5: 1
6: 6
7: 83
Max even #: 456
....
Enter numbers:-5 -10 -44 -2 -27 -9 -27 -9
0:-5
1:-10
2:-44
3:-2
4:-27
5:-9
6:-27
7:-9
Max even #: -2
```

8.5 List nesting

list nesting

Such embedding of a list inside another list is known as list nesting.

Figure 8.5.1: Multi-dimensional lists.

```
my_list = [[10, 20], [30, 40]]
print('First nested list:', my_list[0])
print('Second nested list:', my_list[1])
print('Element 0 of first nested list:', my_list[0][0])
```

```
First nested list: [10, 20]
Second nested list: [30, 40]
Element 0 of first nested list: 10
```

8.5 List nesting

multi-dimensional data structure

List nesting allows for a programmer to also create a multi-dimensional data structure, the simplest being a two-dimensional table, like a spreadsheet or tic-tac-toe board.

Figure 8.5.2: Representing a tic-tac-toe board using nested lists.

```
tic_tac_toe = [
    ['X', 'O', 'X'],
    [' ', 'X', ' '],
    ['O', 'O', 'X']
]

print(tic_tac_toe[0][0], tic_tac_toe[0][1], tic_tac_toe[0][2])
print(tic_tac_toe[1][0], tic_tac_toe[1][1], tic_tac_toe[1][2])
print(tic_tac_toe[2][0], tic_tac_toe[2][1], tic_tac_toe[2][2])
```

```
X O X
  X
O O X
```

8.6 List slicing

- **Slice notation**: A programmer can use slice notation to read multiple elements from a list, creating a new list that contains only the desired elements.

List slice notation.

```
boston_bruins = ['Tyler', 'Zdeno', 'Patrice']
print('Elements 0 and 1:', boston_bruins[0:2])
print('Elements 1 and 2:', boston_bruins[1:3])
```

```
Elements 0 and 1: ['Tyler', 'Zdeno']
Elements 1 and 2: ['Zdeno', 'Patrice']
```

8.6 List slicing

- **stride**: An optional component of slice notation is the stride, which indicates how many elements are skipped between extracted items in the source list.

Table 8.6.1: Some common list slicing operations.

Operation	Description	Example code	Example output
my_list[start:end]	Get a list from start to end (minus 1).	`my_list = [5, 10, 20]` `print(my_list[0:2])`	`[5, 10]`
my_list[start:end:stride]	Get a list of every stride element from start to end (minus 1).	`my_list = [5, 10, 20, 40, 80]` `print(my_list[0:5:3])`	`[5, 40]`
my_list[start:]	Get a list from start to end of the list.	`my_list = [5, 10, 20, 40, 80]` `print(my_list[2:])`	`[20, 40, 80]`
my_list[:end]	Get a list from beginning of list to end (minus 1).	`my_list = [5, 10, 20, 40, 80]` `print(my_list[:4])`	`[5, 10, 20, 40]`
my_list[:]	Get a copy of the list.	`my_list = [5, 10, 20, 40, 80]` `print(my_list[:])`	`[5, 10, 20, 40, 80]`

8.8 List comprehensions

- **list comprehension**: The Python language provides a convenient construct, known as list comprehension, that iterates over a list, modifies each element, and returns a new list consisting of the modified elements.

List comprehension example: A first look.

```
my_list = [10, 20, 30]
list_plus_5 = [(i + 5) for i in my_list]

print('New list contains:', list_plus_5)
```

```
New list contains: [15, 25, 35]
```

8.8 List comprehensions

Table 8.8.1: List comprehensions can replace some for loops.

Num	Description	For loop	Equivalent list comprehension	Output of both programs

1	Add 10 to every element.	`my_list = [5, 20, 50]`
		`for i in range(len(my_list)):`
		`  my_list[ i ] += 10`
		`print(my_list)`
			`my_list = [5, 20, 50]`
			`my_list = [(i+10) for i in my_list]`
			`print(my_list)`
				`[15, 30, 60]`

2	Convert every element to a string.	`my_list = [5, 20, 50]`
		`for i in range(len(my_list)):`
		`  my_list[ i ] = str(my_list[ i ])`
		`print(my_list)`
			`my_list = [5, 20, 50]`
			`my_list = [str(i) for i in my_list]`
			`print(my_list)`
				`['5', '20', '50']`

3	Convert user input into a list of integers.	`inp = input('Enter numbers:')`
		`my_list = []`
		`for i in inp.split():`
		`  my_list.append(int(i))`
		`print(my_list)`
			`inp = input('Enter numbers:')`
			`my_list = [int(i) for i in inp.split()]`
			`print(my_list)`
				`Enter numbers: 7 9 3`
				`[7, 9, 3]`

4	Find the sum of each row in a two-dimensional list.	`my_list = [[5, 10, 15], [2, 3, 16], [100]]`
		`sum_list = []`
		`for row in my_list:`
		`  sum_list.append(sum(row))`
		`print(sum_list)`
			`my_list = [[5, 10, 15], [2, 3, 16], [100]]`
			`sum_list = [sum(row) for row in my_list]`
			`print(sum_list)`
				`[30, 21, 100]`

5	Find the sum of the row with the smallest sum in a two-dimensional table.	`my_list = [[5, 10, 15], [2, 3, 16], [100]]`
		`sum_list = []`
		`for row in my_list:`
		`  sum_list.append(sum(row))`
		`min_row = min(sum_list)`
		`print(min_row)`
			`my_list = [[5, 10, 15], [2, 3, 16], [100]]`
			`min_row = min([sum(row) for row in my_list])`
			`print(min_row)`
				`21`

8.12 Sorting Lists

- **sort()**: One of the most useful list methods is sort(), which performs an in-place rearranging of the list elements, sorting the elements from lowest to highest.
- **sorted()**: The sorted() built-in function provides the same sorting functionality as the list.sort() method, however, sorted() creates and returns a new list instead of modifying an existing list.

Figure 8.9.2: Using sorted() to create a new sorted list from an existing list without modifying the existing list.

```
numbers = [int(i) for i in input('Enter numbers: ').split()]

sorted_numbers = sorted(numbers)

print('\nOriginal numbers:', numbers)
print('Sorted numbers:', sorted_numbers)
```

```
Enter numbers: -5 5 -100 23 4 5
Original numbers: [-5, 5, -100, 23, 4, 5]
Sorted numbers: [-100, -5, 4, 5, 5, 23]
```

8.12 Dictionaries

- **dict**: The dict type implements a dictionary in Python.
- **dictionary comprehension**: The second approach uses dictionary comprehension, which evaluates a loop to create a new dictionary, similar to how list comprehension creates a new list.
- **dict()**: Other approaches use the dict() built-in function, using either keyword arguments to specify the key-value pairs or by specifying a list of tuple-pairs.

8.13 Dictionary methods

- **dictionary method**: A dictionary method is a function provided by the dictionary type (dict) that operates on a specific dictionary object.

Table 8.13.1: Dictionary methods.

Dictionary method	Description	Code example	Output

my_dict.clear()	Removes all items from the dictionary.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`my_dict.clear()`
		`print(my_dict)`
			`{}`

my_dict.get(key, default)	Reads the value of the key from the dictionary. If the key does not exist in the dictionary, then returns default.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`print(my_dict.get('Jane', 'N/A'))`
		`print(my_dict.get('Chad', 'N/A'))`
			`42`
			`N/A`

my_dict1.update(my_dict2)	Merges dictionary my_dict1 with another dictionary my_dict2. Existing entries in my_dict1 are overwritten if the same keys exist in my_dict2.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`my_dict.update({'John': 50})`
		`print(my_dict)`
			`{'Ahmad': 1,`
			`'Jane': 42,`
			`'John': 50}`

my_dict.pop(key, default)	Removes and returns the key value from the dictionary. If key does not exist, then default is returned.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`val = my_dict.pop('Ahmad')`
		`print(my_dict)`
			`{'Jane': 42}`

8.14 Iterating over a dictionary

- **hash**: A hash is a transformation of the key into a unique value that allows the interpreter to perform very fast lookup.
- **view object**: A view object provides read-only access to dictionary keys and values.

dict.items()

```
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda, calories in num_calories.items():
    print(f'{soda}: {calories}')
```

```
Coke: 90
Coke_zero: 0
Pepsi: 94
```

dict.keys()

```
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda in num_calories.keys():
    print(soda)
```

```
Coke
Coke_zero
Pepsi
```

dict.values()

```
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda in num_calories.values():
    print(soda)
```

```
90
0
94
```

REMINDER

UNIVERSITY
OF MIAMI

COLLEGE OF ARTS AND SCIENCES
COMPUTER SCIENCE

Chapter 7 & 8 Assignments in the zyBook

CSC115 - PYTHON PROGRAMMING

UNIVERSITY
OF MIAMI

COLLEGE OF ARTS AND SCIENCES
COMPUTER SCIENCE

Instructor: Dr. Hien Nguyen