HO CHI MINH CITY UNIVERSITY OF TRANSPORT
FACULTY OF INFORMATION TECHNOLOGY
SOFTWARE ENGINEERING DEPARTMENT

CHAPTER 9
STRINGS AND LISTS

CONTENTS

Introduction

- Data Structures in Python:
  - **Strings**
  - **Lists**
  - Dictionaries
  - Tuples
  - Sets

1. Strings

- A string is a sequence of characters. Python treats characters and strings the same way.
- Python has a number of powerful features for manipulating strings:
  - Creating Strings
  - String Indexing
  - String Slicing
  - Modifying Strings
  - Iterating a String
  - String Operators
  - String Functions

Creating Strings

- **Creating strings** as follows:

```
s1 = str()     # Create an empty string object
s2 = str("Welcome") # Create a string object for Welcome
```

- Or

```
s1 = ""# Same as s1 = str()
s2  = "Welcome"# Same as s2 = str("Welcome")
```

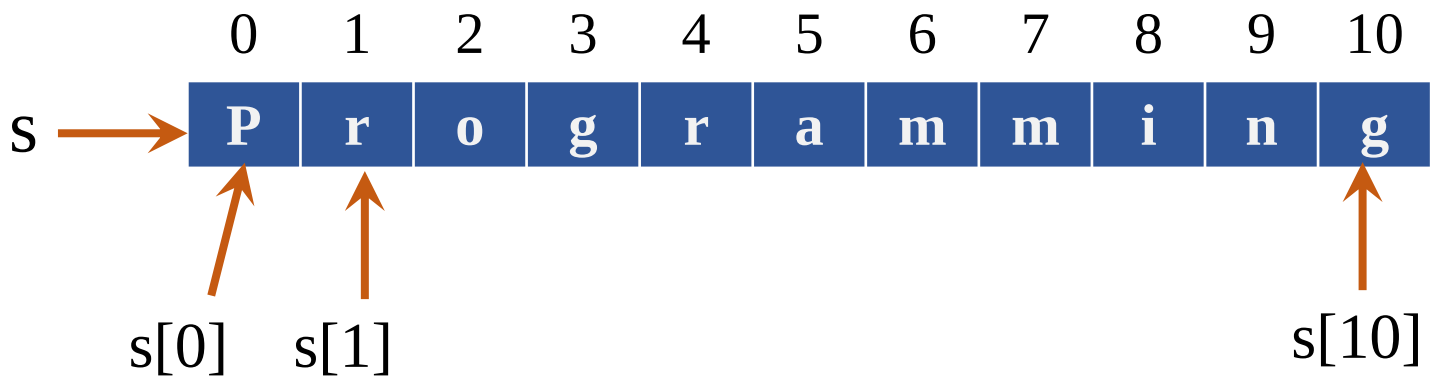
- **A string object is immutable**: once it is created, its content cannot be changed.

String Indexing

- A character in the string can be accessed through the index operator using the syntax:

  s[<index>]

- For example: s = “Programming”



String Indexing

- The indexes are 0 based:

  indexes range from 0 to len(s) – 1

```python
s = "Programming"
print(s[0])
print(s[1])
print(len(s))
print(s[len(s)-1])
#print(s[len(s)]) --> IndexError: string index out of range
```

```
P
r
11
g
```

String Indexing

- Python also allows the use of negative numbers as indexes → indexing occurs from the end of the string backward:

-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
P	**r**	**o**	**g**	**r**	**a**	**m**	**m**	**i**	**n**	**g**
0	1	2	3	4	5	6	7	8	9	10

```python
s = "Programming"
print(s[-len(s)])
print(s[-1])
```

```
P
g
```

- **The actual position is obtained by adding the length of the string with the negative index.**

String Slicing

- The slicing operator returns a slice of the string using the syntax:

s[start : end : step]

- The slice is a substring from index **start** to index **end – 1.**
- The *starting index or ending index may be omitted*. In this case, by default the starting index is 0 and the ending index is the last index

```
s = "Programming"
print(s[1:4])
print(s[:4])
print(s[4:])
print(s[1:-1])
print(s[3:1]) # the slice is empty
```

```
rog
Prog
ramming
rogrammin
```

Modifying Strings

- Can we change individual characters of a string?
- For example:

```
s = "Programming"
s[3] = "X"
```

→ Error

- **Solution**: build a new string from s and reassign it to s

```
s = "Programming"
s = s[:3] + "X" + s[4:]
print(s)
```

```
ProXramming
```

Iterating a String

- A string is iterable → can use a for loop to traverse all characters in the string sequentially.

```
s = "Programming"
for ch in s:
    print(ch,end='-')
```

```
P-r-o-g-r-a-m-m-i-n-g-
```

```
# displays characters at odd-numbered positions
s = "Programming"
for i in range(1,len(s),2):
    print(s[i],end='-')
```

```
r-g-a-m-n-
```

String Operators

- **Concatenation (+):** combines two strings
- **Repetition (*) :** repeats a string a certain number of times
- **Substring Testing** (**in** and **not in**): to test whether a string is in another string

```python
s1 = "Welcome"
s2 = "Python"
s = s1 + " to " + s2
print(s)
```

```
Welcome to Python
```

```python
s = input("Enter a string:")
if "Python" in s:
    print("Python is in",s)
else:
    print("Python is in",s)
```

```
Enter a string:Python is fun
Python is in Python is fun
```

String Operators

- **Comparing Strings**
  - Using the comparison operators (==,!=, >, >=, <,<=)
  - Python compares strings by comparing their corresponding characters, and it does this by evaluating the characters' numeric codes

```python
s1 = "apple"
s2 = "avocado"
print(s1 == s2)
print(s1 < s2)
print(s1 <= s2)
```

```
False
True
True
```

```python
"ab" > "abc"
```

```
False
```

String Functions

Built-in String Functions

- **ord(ch)**: returning the ASCII code for the character ch.
- **chr(code)**: returning the character represented by the code

```python
print(ord('A')) # --> 65
print(chr(98)) # --> 'b'

offset = ord('a') - ord('A')
print(offset) # --> 32
lower = 'z'
upper = chr(ord(lower) - offset)
print(upper) # --> 'Z'

```

```
65
b
32
Z
```

String Functions

Case Conversion

- **s.lower()**: return a copy of string in lowercase
- **s.upper():** return a copy of string in uppercase
- **s.capitalize():** returns a copy of string with the first character converted to uppercase.
- **s.title():** returns a copy of this string with the first letter capitalized in each word.
- **s.swapcase():** Returns a copy of this string in which lowercase letters are converted to uppercase and uppercase to lowercase

```
s1 = "Welcome to Programming"
print(s1.lower())
print(s1.upper())
print(s1.swapcase())

```

```
welcome to programming
WELCOME TO PROGRAMMING
wELCOME TO pROGRAMMING
```

String Functions

Stripping characters from a String

- **s.strip([<char>])**: removes any character from the beginning or the end
  - By default, char is **None** --> ***whitespace characters are removed***

```
s = " Welcome to Python \n"
s.strip()
```

```
'Welcome to Python'
```

```
s = "***10***"
s.strip('*')
```

```
'10'
```

String Functions

Find and Replace

- **s.count(sub[,start[,end]])**: Returns the number of non-overlapping occurrences of this substring
- **s.endswith(suffix[,start[,end]])**: Returns **True** if the string ends with the specified suffix.
- **s.startswith(prefix[,start[,end]])**: Returns **True** if the string starts with the specified prefix.

```python
s = "Welcome to Python"
print(s.count("o"))
print(s.count("o",7))
```

```
3
2
```

```python
print(s.endswith("thon"))
print(s.startswith("good"))
```

```
True
False
```

String Functions

Find and Replace

- **s.find(sub[,start[,end]]):** returns the lowest index in s where substring is found, otherwise return -1
- **s.replace(old, new):** Returns a new string that replaces all the occurrences of the old string with a new string

```
print(s.find("come"))
print(s.find("become"))
```

```
3
-1
```

```
s1 = s.replace("Python","Java")
print(s1)
```

```
Welcome to Java
```

String Functions

Character Classification

- **s.isalnum()**: returns **True** if s is non-empty and all its characters are alphanumeric (either a letter or a number)
- **s.isalpha()**: returns **True** if s is non-empty and all its characters are alphabetic
- **s.isdigit()**: returns **True** if s is non-empty and all its characters are numeric digits

```python
s = "python3"
print(s.isalnum())
```

```
True
```

```python
s = "abc"
print(s.isalpha())
```

```
True
```

```python
s = "2019"
print(s.isdigit())
```

```
True
```

String Functions

Character Classification

- **s.islower()**: returns **True** if s is non-empty and all the alphabetic characters it contains are lowercase.
- **s.isupper()**: returns **True** if s is non-empty and all the alphabetic characters it contains are uppercase

```python
s = "Hello"
print(s.islower())
print(s.isupper())
```

```
False
False
```

String Functions

Split

- **s.split(sep=None, maxsplit=-1)**: Splits a string into a list of substrings.
  - **Sep**: delimiter according which to split the string, default value is **None** (whitespace)
  - **Maxsplit**: maximum number of splits to do, default value is -1 (no limit)

```python
s = "This is split function"
items = s.split()
print(items)
```

```
['This', 'is', 'split', 'function']
```

```python
date = "20/11/2011"
items_1 = date.split(sep='/')
print(items_1)

items_2 = date.split(sep='/',maxsplit=1)
print(items_2)
```

```
['20', '11', '2011']
['20', '11/2011']
```

2. Lists

- Suppose that you need to read 100 numbers, compute their average, and then find out how many of the numbers are above the average.
- How do you solve this problem?
- Create 100 variables??? → This way is impractical.

Lists

Lots of data

How can we store this much data in Python?

We would need 5x19 variables...

→The solution is to store all values together in a **list.**

#	Country	G	S	B	Total
1	Norway	14	14	11	39
2	Germany	14	10	7	31
3	Canada	11	8	10	29
4	United States	9	8	6	23
5	Netherlands	8	6	6	20
6	Sweden	7	6	1	14
7	South Korea	5	8	4	17
8	Switzerland	5	6	4	15
9	France	5	4	6	15
10	Austria	5	3	6	14
11	Japan	4	5	4	13
12	Italy	3	2	5	10
13	Olympic Athletes from Russia	2	6	9	17
14	Czech Republic	2	2	3	7
15	Belarus	2	1	0	3
16	China	1	6	2	9
17	Slovakia	1	2	0	3
18	Finland	1	1	4	6
19	Great Britain	1	0	4	5

Lists

- **A list is a ordered sequence of values**. The values that make up a list are called its **elements**, or its **items**.
- List is a collection data type.
- The elements of a list can be any type.
- Common operations for manipulating lists:
  - Creating Lists
  - List Indexing
  - List Slicing
  - Modifying List
  - Traversing a List
  - List Operators
  - List Functions
  - List Comprehensions
  - Copying Lists

Creating Lists

- The elements in a list are separated by commas and are enclosed by a pair of brackets []
- A list can contain the elements of the same type or mixed types.

```python
empty_list = [] # empty list
print(empty_list)
gold = [14,14,11]
print(gold)
countries = ["Norway","Germany","Canada"]
print(countries)
```

```
[]
[14, 14, 11]
['Norway', 'Germany', 'Canada']
```

Creating Lists

```python
#mix of different types
mix_list = [2,"three",4]
print(mix_list)

#make a list of lists
hands = [['J','Q','K'],['2','2','2'],['6','A','K']]
print(hands)
```

```
[2, 'three', 4]
[['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]
```

```python
# Create a list that consists integer 0 -> n-1
n = int(input("n = "))
mylist = list(range(n))
print(mylist)
```

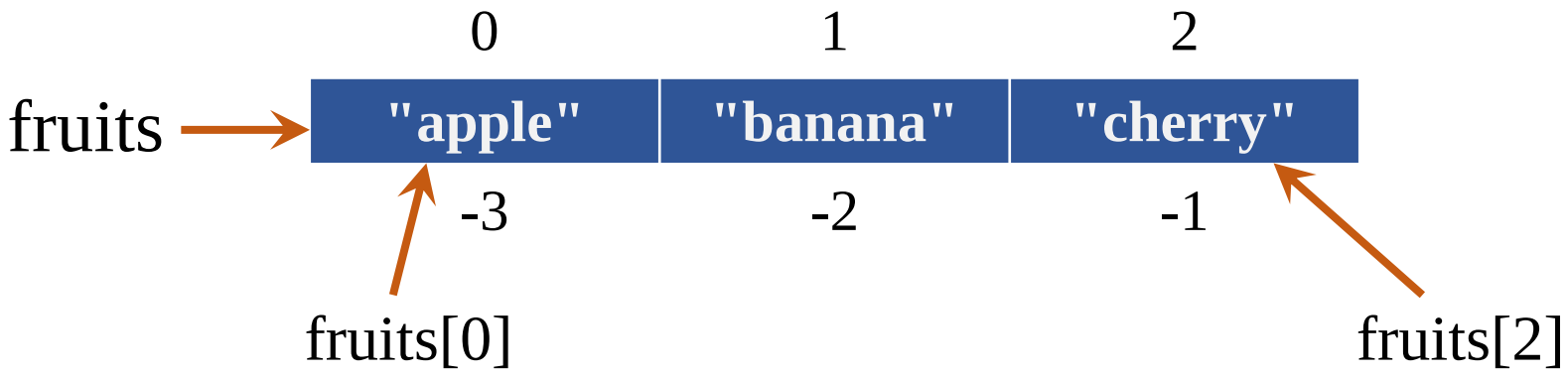
```
n = 10
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Indexing

- We can access the elements of a list using an integer index.

myList[index]

- For example, fruits= ["apple", "banana", "cherry"]




```python
fruits= ["apple","banana","cherry"]
print(fruits[0])
print(fruits[-1])
```

```
apple
cherry
```

List Indexing

- A list can contain sublists.
- To access the elements in a sublist, simply append an additional index:

```
hands = [['J','Q','K'],['2','2','2'],['6','A','K']]
print(hands[0])
print(hands[0][1])
print(hands[2][1])
```

```
['J', 'Q', 'K']
Q
A
```

List Slicing

- The slicing operator returns a slice of the list using the syntax:

  mylist[start : end]

- The slice is a sublist from index **start** to index **end – 1**

```
primes = [2, 3, 5, 7,11,13]
print(primes[2 : 4])
print(primes[:2]) # two first primes
print(primes[3:]) # all the primes from index 3
print(primes[:]) # all the primes
print(primes[1:-1]) # All the primes except the first and last
```

```
[5, 7]
[2, 3]
[7, 11, 13]
[2, 3, 5, 7, 11, 13]
[3, 5, 7, 11]
```

Modifying List

- Lists are "mutable", meaning we can change their elements.
- To change the value of a specific element, refer to the index number:

```
fruits= ["apple","banana","cherry"]
fruits[1] = "grapes"
print(fruits)
```

```
['apple', 'grapes', 'cherry']
```

```
fruits[1:] = ["kiwi","durian"]
print(fruits)
```

```
['apple', 'kiwi', 'durian']
```

```
fruits[1:] = [] # remove elements from 1 to last
print(fruits)
```

```
['apple']
```

Traversing a List

- Using **for** loop, which enables you to traverse the list sequentially without using an index variable.

```
fruits= ["apple","banana","cherry"]
for f in fruits:
    print(f,end=" ")
```

```
apple banana cherry
```

- Use an index variable if you wish to traverse the list in a different order or change the elements in the list.

```
fruits= ["apple","banana","cherry"]
size = len(fruits)
for i in range(size-1, -1, -1): # i = size-1 --> 0
    print(fruits[i],end=" ")
```

```
cherry banana apple
```

Traversing a List

```python
numbers = [4,2,8,7]
s = 0
for num in numbers:
    s = s + num
print("Sum all elements in list: ",s)
```

```
Sum all elements in list:  21
```

```python
# Print all positive elements in list
numbers = eval(input("Enter a list of integer: "))
flag = False
for num in numbers:
    if num > 0:
        flag = True
        print(num)
if not flag:
    print("The list has no positive elements!")
```

```
Enter a list of integer: [-9,0,4,-2,5]
4
5
```

List Operators

Operator	Description
in	True if element x is in list.
not in	True if element x is not in list.
+	Concatenates list1 and list2.
*	n copies of list concatenated.

```
countries = ["Norway","Germany","Canada"]
"Canada" in countries
```

```
True
```

```
list1 = [1,2,3]
list2 = [4,5,6]
list3 = list1 + list2
print(list3)
print(list1*2)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3]
```

List Operators

- **Comparing Lists**
  - Using the comparison operators (>, >=, <, <=, ==,!=)
  - The comparison uses ordering: the first two elements are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two elements are compared, and so on.

```
color1 = ["green", "red", "blue"]
color2 = ["red", "blue", "green"]
list1 = [1,2,3]
print(color1 != color2)
print(color1 > color2)
#color1 > list1 --> Error
```

```
True
False
```

List Functions

- **mylist.append(<object>)**: appends object to the end of list

```
mylist = [5, 3, 4, 1, 6, 4]
mylist.append(10)
print(mylist)
```

```
[5, 3, 4, 1, 6, 4, 10]
```

```
# Create a list with n elements
n = int(input("Enter number of elements in list: "))
mylist = []
for i in range(n):
    value = eval(input(f"mylist[{i}] = "))
    mylist.append(value)
print("My list is", mylist)
```

```
Enter number of elements in list: 3
mylist[0] = 6
mylist[1] = 1
mylist[2] = 2
My list is [6, 1, 2]
```

List Functions

- **mylist.extend(<iterable>)**: Extend list by appending elements from the iterable, behaves like the + operator.

```
list1 = [2,3,4]
list1.extend([5,6])
print(list1)
list1.extend("Python")
print(list1)
```

```
[2, 3, 4, 5, 6]
[2, 3, 4, 5, 6, 'P', 'y', 't', 'h', 'o', 'n']
```

- **mylist.count(<object>)**: Returns the number of times element <object> appears in the list

```
list1 = [3,1,4,5,4]
list1.count(4)
```

```
2
```

List Functions

- **mylist.index(<object>)**: Returns the index of the first occurrence of element <object> in the list.

```
list1 = [3,4,1,5,4]
list1.index(4)
```

```
1
```

- **mylist.insert(<index>, <object>)**: Inserts an element <object> at a given index.

```
list1 = [1,5,7,9]
list1.insert(1,3)
print(list1)
```

```
[1, 3, 5, 7, 9]
```

List Functions

- **mylist.remove(<object>)**: Removes the first occurrence of element x from the list.

```
list1 = [3,4,1,5,4]
list1.remove(4)
print(list1)
#list1.remove(6) --> Error: 6 not in list
```

```
[3, 1, 5, 4]
```

- **mylist.pop(index = -1)**: Removes the element at the given position and returns it.

```
list1 = [3,4,1,5,4]
list1.pop(1) # return 4
print(list1)
list1.pop()
print(list1)
```

```
[3, 1, 5, 4]
[3, 1, 5]
```

List Functions

- **mylist.reverse()**: Reverses the elements in the list.

```
odd = [1,3,5,7,9]
odd.reverse()
print(odd)
```

```
[9, 7, 5, 3, 1]
```

- **mylist.sort()**: Sorts the elements in the list in ascending order.

```
odd = [5,1,7,3,9]
odd.sort()
print(odd)
```

```
[1, 3, 5, 7, 9]
```

List Comprehensions

- Provide a concise way to create a sequential list of elements.

```python
squares = []
for i in range(5):
    squares.append(i*i)
print(squares)
```

```
[0, 1, 4, 9, 16]
```

→

```python
squares = [i*i for i in range(5)]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

- Syntax:

new_list = [expression for member in iterable (if conditional)]

  - A list comprehension consists of brackets containing an expression followed by a **for** clause, then zero or more **for** or **if** clauses.

List Comprehensions

Example 1

```
numbers = [4,1,3,5,2,6]
even = [x for x in numbers if x%2 == 0]
print(even)
```

```
[4, 2, 6]
```

Example 2

```
fruits = ["apple","avocado","banana","cherry","watermelon"]
short_fruits = [f for f in fruits if len(f) <= 6]
print(short_fruits)
```

```
['apple', 'banana', 'cherry']
```

Example 3

```
fruits = ["apple","avocado","banana","cherry","watermelon"]
short_fruits = [f.upper() for f in fruits if len(f) <= 6]
print(short_fruits)
```

```
['APPLE', 'BANANA', 'CHERRY']
```

Copying Lists

- Using the assignment statement (=) to copy the data in one list to another list → **Result???**

```python
list1 = [1,3,5,7]
list2 = list1
print("Before appending:",end=" ")
print(len(list1))
list2.append(9)
print("After appending:",end=" ")
print(len(list1))
print("After modifying")
list1[2] = 11
print(list2)
```

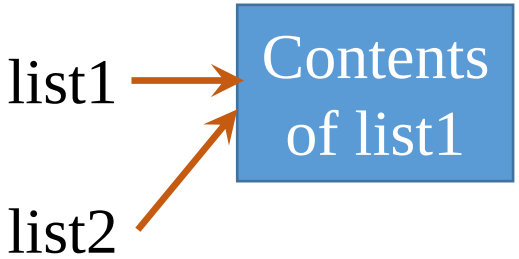
```
Before appending: 4
After appending: 5
After modifying
[1, 3, 11, 7, 9]
```

```python
list1 = [1,3,5,7]
list2 = [1,3,5,7]
print("Before appending:",end=" ")
print(len(list1))
list2.append(9)
print("After appending:",end=" ")
print(len(list1))
print("After modifying")
list1[2] = 11
print(list2)
```

```
Before appending: 4
After appending: 4
After modifying
[1, 3, 5, 7, 9]
```

Copying Lists

- list2 = list1 → Do not copy the contents
- So, get a duplicate copy of list1 into list2,



you have to copy individual elements from the source list to the target list using list comprehension or operator + or copy() function

```
list1 = [1,3,5,7]

list2 = [x for x in list1]

print(list2)
print(list2 is list1)
```

```
[1, 3, 5, 7]
False
```

Or simply

```
list1 = [1,3,5,7]

list2 = [] + list1

print(list2)
print(list2 is list1)
```

```
[1, 3, 5, 7]
False
```