COMP90059 Semester 1, 2022
Introduction to Programming

THE UNIVERSITY OF
MELBOURNE

Lecture 4: Sequences and Strings

Dr Simon D'Alfonso

School of Computing and Information Systems
Faculty of Engineering and Information Technology

Lecture Overview

- Sequences
- String access, manipulation and methods
- A glimpse into *lists*
- A glimpse into *iteration* and *for* loops

3 and 4 == ?

```
>>> a = 3
>>> b = 4
>>> a and b
4
>>> a or b
3
>>> not a and b
False
```

https://stackoverflow.com/questions/49658308/how-does-the-logical-and-operator-work-with-integers

Solutions to Lecture 3 Challenges

1. Write code to determine if a given number *x* is even
2. Write code to calculate the ceiling of a given number *x*, without using math.ceil()
3. Write code to calculate the area of a circle with a given radius *r*

What we have looked at so far

- The two main things:
  - Variables
  - Conditionals

```
#Example
string1 = "This is a string"
if isinstance(string1, str):
    print(string1)
```

Next up

But given a string, how do we?

- Reverse a string
- Find the first occurrence of the character ‘a’
- Return the third character in a string
- Convert all characters to uppercase

String Manipulations and Methods

- Access individual characters in a string
- Retrieve a substring from a string
- Search for a substring in a string
- Using libraries for string manipulations
- Splitting a string into a list that can be manipulated

Data Structures

- Data structure - refers to methods of organizing units of data within larger datasets. Achieving and maintaining specific data structures helps to improve data access and value.
- In Python, sequences are a generic term for an ordered set which means that the order in which we input the items will be the same when we access them.

Strings

- A string is a data structure.
- A string is a sequence of zero or more characters
- A string is an immutable data structure. That means, its internal data elements, the characters, can be accessed, but the individual elements cannot be modified
- Lists, which we will very briefly look at later in this lecture, are data structures that are sequences but mutable.

Structure of Strings

- We sometimes might want to access or inspect characters at a given position without needing to visit the entire string (part of a string - substring).
- The string is arranged as shown. There is an index that helps us step through or inspect a position.

H	i		T	h	e	r	e
0	1	2	3	4	5	6	7

String Indexing

To access a position in a string and return its content, we use the following syntax, where *x* is an integer and [] is known as the subscript operator:

string[*x*]

Examples

```
>>> "abc"[2]
'c'

>>> string1 = "Python"
>>> string1[3]
'h'
```

What about string1[9]?

Negative Indexing

- We have seen what happens when the index is too large. What happens if the index is less than 0? Does it give us an error?
- Negative indices work from the end of the string, so -1 indexes the last character.
- Note the negative indexes are one-offset (i.e., start from -1) while the positive indexes are zero-offset (i.e., start from 0).

```
>>> string1 = "Python"
>>> string1[-2]
'o'
```

String Slicing

- Python’s subscript operator can be used to obtain a substring through a process called slicing - place a colon (:) in the subscript, and an integer value can appear on either side of the colon.
- For example, the following code accesses the substring of the string "the example is on slicing" starting at index 2, up to (but not including) index 10.

```
>>> s = "the example is on slicing"
>>> s[2:10]
'e exampl'
```

Accessing Substrings (slicing)

- The notation 2:10 is known as a slice.
- Remember that a slice starts at the first index but finishes one before the end index. This is consistent with indexing: indexing also starts from zero and goes up to one before the length of the string.
- You can see this by slicing with the value of len():

```
>>> s = "the example is on slicing"
>>> len(s) #len = 25, but 24 is max index
>>> s[0:len(s)]
'the example is on slicing'
```

More on Slicing: Example

You can also slice with negative indices. The same basic rule of starting from the start index and stopping one before the end index applies:

s = "testing slicing 101"

Try:

- s[4:-7]
- s[-7:-1]
- s[-6:len(s)]

More on Slicing

Python provides two shortcuts for common slice values:

- if the start index is 0 then you can leave it blank
- if the end index is the length of the string, then you can leave it blank

s = "testing slicing 101"

Try:

- s[:5]
- s[5:]
- s[:]

Changing the Step Size and Direction

You can specify a third number which indicates how much to step through the list by.

For example, if you want every second element you can do this:

```
>>> s = "abcdef"
>>> s[::2]
'ace'
```

Or you can specify a range with a step:

```
>>> s = "abcdef"
>>> print(s[0:3:2])
'ac'
```

Changing the Step Size and Direction

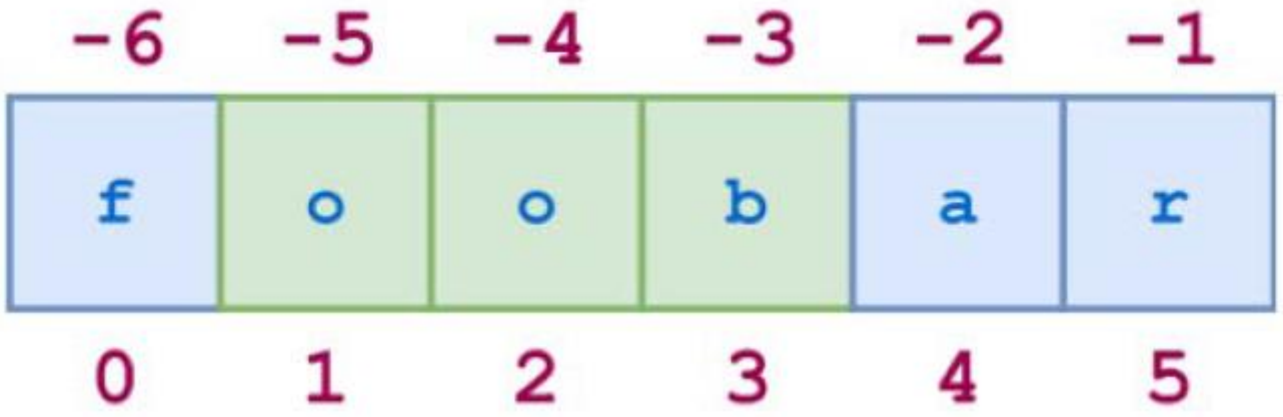
If this third parameter (step) is negative this indicates that the direction of the steps will be right-to-left:

```
>>> s = "abcdef"
>>> s[2::-1]
'cba'
>>> s[2:0:-1]
'cb'
>>> s[-4:-6:-1]
'cb'
```

Note that the direction of the indices must be changed also. If there is nothing in between the indices, an empty string is returned (unlike indexing beyond the ends of the string, which led to an error):

```
>>> s = "abcdef"
>>> s[0:2:-1]
?
```

Exercise 1: Accessing characters and substrings



Using indexing and slicing, write expressions to get the following from the string 'foobar':

1. The character 'a'
2. The substring 'foo'
3. The substring 'bar'
4. The string 'foa'
5. The string 'rab'

Exercise 1 Solutions

1. "foobar"[4]
2. "foobar"[0:3]
3. "foobar"[3:]
4. "foobar"[0:6:2]
5. "foobar"[-1:-4:-1]

Exercise 2: Palindrome Test

A palindrome is a string that is the same forwards as backwards:

- dad
- radar
- refer

Write some code that checks whether a given string is a palindrome or not.

Exercise 2: Solution

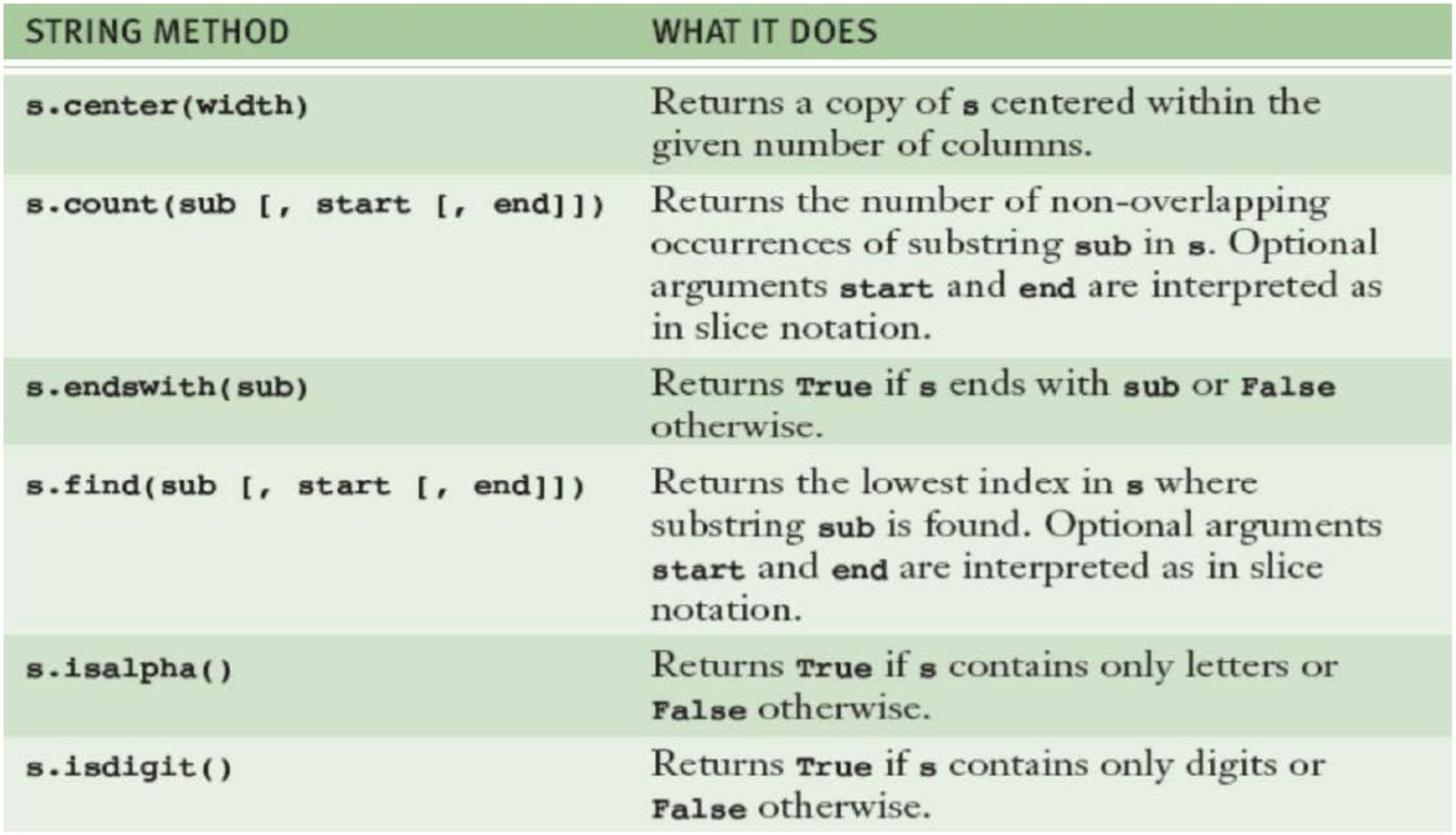
```
string1 = "civic"

if string1 == string1[::-1]:
    print("Is palindrome")
else:
    print("Is not palindrome")
```

String methods

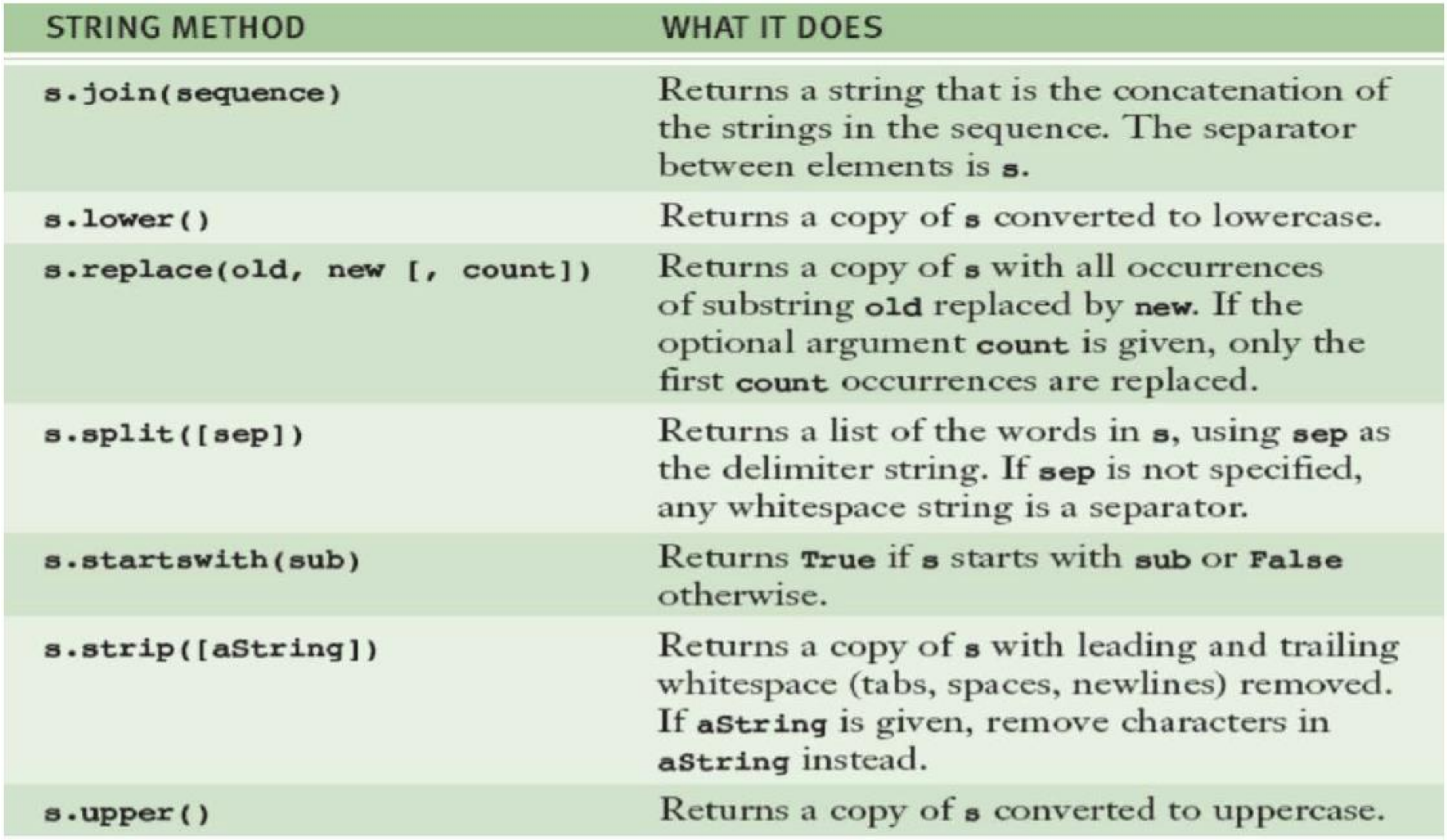
- Methods can expect arguments and return values
- A method knows about the internal state of the object with which it is called
- In Python, all data values are objects
- dir(str) or help(str) will give you the entire list of string methods or library functions.
- https://www.w3schools.com/python/python_ref_string.asp

Some String Methods



STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

Some More String Methods



STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

String Method Examples

s = "The quick brown fox jumps over the lazy dog" #pangram

Try:

- s.upper()
- s.startswith('The quick')
- s.split()
- s.find('fox')
- s.replace('lazy', 'tired')
- s.isupper()

List() method

The method list() takes sequence types such as strings and converts them to lists:

list(seq)

```
>>> num = "123"
>>> list(num)
['1', '2', '3']

>>> word = "hello"
>>> list(word)
['w', 'o', 'r', 'd']
```

The *in* operator

x in y - returns True if x is present in y, False otherwise

Strings:

```
>>> "red" in "predict"
True
```

Lists:

```
>>> 1 in [1, 2, 3]
True
>>> 'a' in ['a', 'b', 'c’]
True
>>> 'dog' in ['dog', 'cat', 'rabbit']
True
```

Mutability and Immutability

Strings are immutable, lists are mutable

```
>>> string1 = "ace"
>>> list1 = list(string1)
>>> string1[2] = "t"
>>> list1[2] = "t"
```

Exercise 3: Deciding if a password is secure

An organization defines a valid password as follows:

- It must be 8 – 15 characters in length
- It must contain at least one uppercase character (A-Z)
- It must contain at least one lowercase character (a-z)
- It must contain at least one digit

Write a program that takes a password string input, checks it according to the rules above, and prints “valid password” or “invalid password” depending on the result.

Iteration and *for* loops

- Iteration is a core technique in which a block of code is repeated in a loop
- *for* loops are one type of iteration structure

```
for <variable> in <iterable>:
    <statement>
    <statement>
    …
```

Using for loops with strings

```
pwd = "Fd4x32?"
for char in pwd: #print out each character on a new line
    print(char)

#---

#convert all lowercases to uppercase and all uppercase to
lowercase
new_pwd = ""
for char in pwd:
    if char.isupper():
        new_pwd = new_pwd + char.lower()
    elif char.islower():
        new_pwd = new_pwd + char.upper()
    else:
        new_pwd = new_pwd + char

print(new_pwd)
```

Exercise 3 Alternative Solution

Using a *for* loop greatly simplifies things:

```
has_lower = False
has_upper = False
has_digit = False

if len(pwd) < 8 or len(pwd) > 15:
    print("invalid password")
else:
    for char in pwd:
        if char.islower():
            has_lower = True
        if char.isupper():
            has_upper = True
        if char.isdigit():
            has_digit = True

    if has_lower and has_upper and has_digit:
        print("valid password")
    else:
        print("invalid password")
```

Using for

Sometimes it is necessary to use a loop; it is not merely a simplifying option. This is when there is no specified limit on the length of a sequence, and we need to go through the sequence one by one.

Write a program to count how many lower-case letters are in a password.

Summary

Today we covered:

- Sequences
- String access, manipulation and methods
- A glimpse into lists
- A glimpse into iteration and for loops

Lecture 4 Challenges

Write a program to:

- Devowel a given string and print the result. Devowelling is the process of removing vowels (a, e, i, o, u) from a string.
- Determine whether a given string contains a double consecutive occurrence of any characters (e.g., ‘ee’, ‘oo’, ‘11’)

Lecture Identification and Acknowledgement

Coordinator / Lecturer: Simon D'Alfonso

Semester: Semester 1, 2022

These slides include materials from 2020 - 2021 instances of COMP90059 run by Kylie McColl or Wally Smith