

# COMPSCI 101

## Principles of Programming

### Lecture 5 – Manipulating strings, string methods, dot notation

## Learning outcomes

At the end of this lecture, you will know how to:

- use dot notation (apply string methods to string instances)
- use string methods such as `upper()`, `lower()`, `strip()`, `find()`, `rfind()`
- use the inbuilt functions: `min()`, `max()`, `round()`, `abs()`

## Recap

### From lecture 4

Use the `len()` function to calculate how many characters are in a string

Obtain a single character from a string

Slice strings

Concatenate strings

```
words = " Prince Charming "
length = len(words)

letter1 = words[3]
letter2 = words[-5]
letter3 = words[len(words) - 2]

letters1 = words[3:6]
letters2 = words[:6]
letters3 = words[6:]
letters4 = words[-3:]

word = letter1 + letter2
word = word + " " + letter3

print(letters1, letters2, letters3, letters4, word)
```

## String Slicing – The Step Option

When slicing a string an optional last argument is the step:

`a_string[start: end: step]`

When the step option is omitted the default step is 1

The step option dictates which characters will form the string slice, e.g.

```
a_string = "0123456789"
print(a_string[2: 7: 2]) #prints the slice "246"
a_string = "Sing for your supper"
print(a_string[5: : 4]) #prints the slice "fy p"
```

Complete the output:

```
a_string = "ABCDEFGHJKLMNOPQRSTUVWXYZ"
print(a_string)
print("1.", a_string[0: 8: 2])
print("2.", a_string[1: 15: 5])
print("3.", a_string[3: : 6])
print("4.", a_string[: 10: 3])
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1.
2.
3.
4.
```

## Dot notation

Every object type, as well as storing some data, has some defined methods which can be applied to that particular type of object.

Variables which reference an object are called instances, e.g. in the following code, `greeting` is a **string instance** and `number` is an **instance of type int**.

```
greeting = "Hello World"
number = 234
```

String instances have many methods which can be applied to them such as `upper()`, `lower()`, `find()`, `strip()`, `isalpha()`, `isdigit()`, `rfind()`, `split()` ... In this lecture we will look at a few of these methods.

In order to apply a method to an object we use **dot notation**, i.e. the variable name, followed by a dot, followed by the method name.

**instance\_name.method\_name(...)**

Note that, methods (like functions) use parentheses (round brackets) after the method name.

## String methods – upper(), lower()

The **upper()** method returns a **new** string object with all the characters converted to upper case. The **lower()** method returns a new string object with all the characters converted to lower case.

For example,

```
greeting = "Hello World"
greeting_lower = greeting.lower()
greeting_upper = greeting.upper()
print(greeting, greeting_lower, greeting_upper)
```

Hello World hello world HELLO WORLD

Notice that there is a total of three string objects

## String methods – find()

The **find()** method is used to look for the position (index number) of the first occurrence (from the left) of some characters. If the characters are found, the `find()` method returns the index number, otherwise the `find()` method returns -1.

`greeting` `010100101`

010100101										
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

For example,

```
greeting = "Hello World"
position1 = greeting.find(" ")
position2 = greeting.find("z")
position3 = greeting.find("orl")
print(position1, position2, position3)
```

5 -1 7

## String methods – rfind()

The **rfind()** method is used to look for the index position of the last occurrence (from the right) of some characters. If the characters are found, the `rfind()` method returns the index number, otherwise the `rfind()` method returns -1.

`greeting` `010100101`

010100101										
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

For example,

```
greeting = "Hello World"
position1 = greeting.find("o")
position2 = greeting.rfind("o")
position3 = greeting.rfind("orl")
position4 = greeting.rfind("lro")
print(position1, position2, position3, position4)
```

4 7 7 -1

## String methods – strip()

The **strip()** method returns a new string object with all white space from the beginning and end of the string removed. It does not remove spaces from inside the string.

For example,

```
letters1 = "  H e l l o oooo  "
letters2 = letters1.strip()

length1 = len(letters1)
length2 = len(letters2)

letters1 = "***" + letters1 + "***"
letters2 = "***" + letters2 + "***"

print(length1, length2, letters1, letters2)
```

```
21 14 ***  H e l l o oooo  *** ***H e l l o oooo***
```

Notice that there are two string objects

## Exercise

Complete the following program so that it prints the initial from the first name followed by a full stop, a space and followed by the surname. Assume the full name is always two names separated by a single space.

```
full_name = "Wystan Auden"

initialled_name = first_letter + ". " + last_name
print(initialled_name)
```

```
W. Auden
```

Funeral Blues: Stop all the clocks, cut off the telephone/ Prevent the dog from barking ...

## Common Python inbuilt functions

**min()** is an **inbuilt function** which can be used to find the smallest number from a comma separated set of numbers and **max()** is the **inbuilt function** which can be used to find the largest number from a comma separated set of numbers, e.g.,

```
num1 = 32
num2 = 16
smallest = min(num1, num2)
print(smallest)

smallest = min(32.7, 56.4, 3, -1.1, 56.99, -1.2)
print(smallest)

largest = max(num1, num2)
print(largest)

largest = max(32.7, 56.4, 3, -1.1, 56.99, -1.2)
print(largest)
```

```
16
-1.2
32
56.99
```

## Common Python inbuilt functions

The **inbuilt function**, **round()**, is used to round numbers to the closest whole number (or rounded to a number of digits after the decimal point), e.g.,

```
num1 = 32.657123
num2 = 16.48926
num3 = -16.48926

print(round(num1))
print(round(num2))
print(round(num3))
print()

print(round(num1, 2))
print(round(num2, 3))
print(round(num3, 4))
```

```
33
16
-16

32.66
16.489
-16.4893
```

## Common Python inbuilt functions

Note that the **function, round()** with a single argument returns an int number and that rounding an int returns the int unchanged,

e.g.

```
print("round(32.657123, 0): ", round(32.657123, 0))
print("round(16.48926, 0): ", round(16.48926, 0))
print("round(32.657123): ", round(32.657123))
print("round(16.48926): ", round(16.48926))
print("round(24.0, 0): ", round(24.0, 0))
print("round(24.0, 1): ", round(24.0, 1))
print("round(24, 0): ", round(24, 0))
print("round(24.0): ", round(24.0))
print("round(24): ", round(24))
```

```
round(32.657123, 0): 33.0
round(16.48926, 0): 16.0
round(32.657123): 33
round(16.48926): 16
round(24.0, 0): 24.0
round(24.0, 1): 24.0
round(24, 0): 24
round(24.0): 24
round(24): 24
```

## Common Python inbuilt functions

The **inbuilt function, abs()**, is used to get the absolute value (the magnitude) of a number (int or float), e.g.

```
num1 = 32
num2 = -32
num3 = abs(16 - 23)

print("1.", abs(num1))
print("2.", abs(num2))
print("3.", num3)
print("4.", abs(-16.78))
```

```
1. 32
2. 32
3. 7
4. 16.78
```

## round() – unexpected result

Sometimes the round() function seems to give an unexpected result e.g.,

```
num1 = 1.5
num2 = 2.5
num3 = 3.5
print(round(num1))
print(round(num2)) #surprising result
print(round(num3))
```

```
2
2
4
```

This problem happens because floating point numbers are stored in a finite space, e.g. 0.1 has an infinite number of digits when converted to base 2

```
0.000110011001100110011001100110011001100110011001100110011001...
```

but, when stored in the computer memory, float numbers are assigned exactly 64 bits of space which means that some of the bits are cut off. If 2.5 is actually stored as 2.4999999999999999...99 then it is rounded to 2 (not the expected 3).

```
round(4.235, 2) gives the number 4.24
round(4.265, 2) gives the number 4.26
```

## Exercise

Complete the following program so that it prints the total tax and the net pay rounded to a whole number. The first \$14000 is not taxed. The next amount up to \$38500 is taxed at 24% and the rest is taxed at 34%.

```
salary = 54000
no_tax_boundary = 14000
rate1_boundary = 38500
rate1 = 0.24
rate2 = 0.34
```

```
Salary: $54000
Amount to be taxed at: 24%: $24500
Tax at rate1: $5880
Amount to be taxed at: 34%: $15500
Tax at rate2: $5270
=====
Total tax: $11150
Net pay: $42850
=====
```

```
#Print the information
print("=====")
print("Total tax: $", total_tax, sep = "")
print()
print("Net pay: $", net_pay, sep = "")
print("=====")
```

## Summary

### In Python:

- use dot notation when using string methods with string instances
- the string methods: upper(), lower(), strip(), find(), rfind() can be used with string instances
- Some Python inbuilt functions are: min(), max(), round()

## Examples of Python features used in this lecture

```
greeting = "Hello World"
position1 = greeting.find("o")
position2 = greeting.rfind("o")
position3 = words.find("Z")
position4 = words.rfind("o W")

greeting_lower = greeting.lower()
greeting_upper = greeting.upper()

smallest = min(32.7, 56.4, 3, -1.1, 56.99, -1.2)
largest = max(32.7, 56.4, 3, -1.1, 56.99, -1.2)

num1 = 32.657123
print(round(num1))
print(round(num1, 2))

num2 = abs(20 - num1)
print(num2)
```