UNSW Business School
COMM5007 Coding for Business

Lecture 2
Python Fundamentals Part 2

Jan 2024

A quick review

- Write Python code
- Data types - Integer, Float, String, Boolean
- Type conversions
- Variables
- Assignment/Print Statement

Reminder on Quiz 1

- Quiz 1 **at the beginning** of Lab 3 on Wed, Jan 10 and Thurs, Jan 11
- What to bring
  - **2B pencil and eraser**
  - Any printed material (Open book, but not open laptop)
  - Student card (tutor will verify)
- Each lab will use different questions, but at the same difficulty level
- The content in lecture 1 and 2 is covered
- No collaboration
- 7 Multiple choice questions
- 15 minutes
- 7.5% of final grade

Copyright

- There are some file-sharing websites that specialise in buying and selling academic work to and from university students.

- If you upload your original work to these websites, and if another student downloads and presents it as their own either wholly or partially, you might be found guilty of collusion — even years after graduation.

- These file-sharing websites may also accept purchase of course materials, such as copies of lecture slides and tutorial handouts. By law, the copyright on course materials, developed by UNSW staff in the course of their employment, belongs to UNSW. It constitutes copyright infringement, if not academic misconduct, to trade these materials.

Country

- UNSW Business School acknowledges the Bidjigal (Kensington campus) and Gadigal (City campus) the traditional custodians of the lands where each campus is located.
- We acknowledge all Aboriginal and Torres Strait Islander Elders, past and present and their communities who have shared and practiced their teachings over thousands of years including business practices.
- We recognize Aboriginal and Torres Strait Islander people’s ongoing leadership and contributions, including to business, education and industry.

UNSW Business School acknowledges
the Bidjigal (Kensington campus) and Gadigal (City campus)
the traditional custodians of the lands
where each campus is located.

UNSW Business School. (2023, August 18). *Acknowledgement of Country* [online video]. Retrieved from https://vimeo.com/369229957/d995d8087f

At UNSW you are free to...

- Respectfully disagree about anything
- Express different opinions
- Write your beliefs
- Show your beliefs
- Leave any club or organisation

UNSW SYDNEY

It's not acceptable to...

- Attempt to censor opinions
- Use hate speech
- Make threats or instil fear
- Make false accusations
- Access or share others private information without consent

We are here to help...

- Tell a teacher
- Tell UNSW Psychology and Wellness
- Report to UNSW Complaints
- Report to UNSW Security
- Report a crime to police

Find out more

Outline

- **Arithmetic Operators**
- Relational Operators
- Logical Operators
- String and Escape Characters
- String Operations

Overview of operators

- **Operators** are special symbols that represent computations like *addition, subtraction, division, and multiplication*.
- The values that an operator is applied to are called **operands**
- `20 + 32` is a computation with
  - operator +
  - two operands 20 and 32
- Types of Operators
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators

Arithmetic Operators

Operator	Operation	Description	Example
+	**Addition**	Adds two values or variables	2 + 3
-	**Subtraction**	Subtracts right hand operand from left hand operand.	5 - 2
*	**Multiplication**	Multiplies values	2 * 3
/	**Division**	Divides left hand operand by right hand operand	5.0 / 2.0
%	**Remainder**	Divides left hand operand by right hand operand and returns the remainder	5 % 2 (result is 1)
**	**Exponentiation**	Performs exponential (power) calculation	2**3 (result is 8)
//	**Floor/Integer Division**	The division of operands where the result is rounded down to the nearest whole number.	9//2 = 4 (9/2 = 4.5) -11//3 = -4 (-11/3 = -3.67)

We can use operators to construct expressions

What is an expression?

- A combination of values, variables, and **operators** that the interpreter can evaluate to produce a value
- Returns a value always
- A value by itself is also considered an expression, and so is a variable

```
17
x
x + 17
```

What operator shall we use to build an expression for this example?

If you have 51 candies and want to share them **equally** between 6 kids

- How many does each kid get? (The result should be integer)
- Is there any candy left? (The result should be boolean)
- How many candies are remaining? (The result should be integer)

Division operator’s result is always of type float

In division, the result would always be a **float** in Python, no matter whether the operands are both integer, float or a combination of integer and float.

```
print(6/2)
print(5/2)
print(6.0/2.0)
print(5.0/2.0)
print(2/6.0)
print(2/6)
```

Operation precedence rules

- When more than one operator appears in an expression, the order of calculation depends on the **rules of precedence**.

- Operation precedence:
    1. The subexpression within parentheses/brackets
    2. Exponentiation (powers and square roots)
    3. Multiplication, division, and remainder, performed from left to right
    4. Addition and subtraction, performed from left to right

Highest

Lowest

E.g.,

```
3 * (1 + 2) / 4**2
```

Outline

- Arithmetic Operators
- **Relational Operators**
- Logical Operators
- String and Escape Characters
- String Operations

Relational Operators (1/2)

Relational operators, i.e., comparison operators, are used to compare the values of two expression.

Syntax	Semantics	Description	Example
==	Equal to	If the values of two operands are equal, then the condition becomes true.	5 == 6 (False) 5 == 5 (True)
!=	Not equal to	If values of two operands are not equal, then condition becomes true.	5 != 6 (True)
>	Greater than	If the value of left operand is greater than the value of right operand, then condition becomes true.	5 > 6 (False) 6 > 5 (True)

Relational Operators (2/2)

Syntax	Semantics	Description	Example
<	Less than	If the value of left operand is less than the value of right operand, then condition becomes true.	6 < 5 (False) 5 < 6 (True)
>=	Greater than or equal to	If the value of left operand is greater than **or** equal to the value of right operand, then condition becomes true.	5 >= 5 (True) 5 >= 4 (True) 5 >= 6 (False)
<=	Less than or equal to	If the value of left operand is less than **or** equal to the value of right operand, then condition becomes true.	5 <= 5 (True) 5 <= 6 (True) 5 <= 4 (False)

Exercise

Try the code below:

```
print (1 > 1)
print (1 >= 1)
print (2 == 1)
print (2 != 1)
print("a" != 1)
```

Outline

- Arithmetic Operators
- Relational Operators
- **Logical Operators**
- String and Escape Characters
- String Operations

Logical Operators

- Logical operators include `and, or, not`
- Sometimes we want to check **more than one condition at once**, e.g., we might want to check **if condition1 and condition2 are both True**
- The semantics (meaning) of these operators is similar to their meaning in English, e.g., `x > 0 and x < 10` is true only if x is greater than 0 ***and*** less than 10.
- **Precedence rules**: not > and > or

How to evaluate logical operators

- The `and` statement is only True when both conditions are true

`and`

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- The `or` statement is True if one condition, or both are True

`or`

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

- The `not` statement outputs the opposite truth value

`not`

x	not x
False	True
True	False

UNSW SYDNEY

Exercise

Try below the statements:

```
a = True
b = True
c = False

print(a and b)
print(a or b)
print(not a)
print(not b)

print(a or b and c)
# Do we go from left to right?
# What's the best way to avoid confusion?
```

Outline

- Arithmetic Operators
- Relational Operators
- Logical Operators
- **String and Escape Characters**
- String Operations

Strings

- A string is a *sequence* of characters, e.g., `'banana'`
- Strings in python are surrounded by single quotation marks, or double quotation marks

```
'hello' or "hello"
```

- A string can include spaces or digits

```
'hello1'
'hello hello'
```

- A string can also include special characters

```
'hello, world'
'hello !!!'
```

- You can display a string literal with the print() function

```
print("Hello")
print('Hello')
```

Single vs. Double Quotes

- The most common use of single and double quotes is to represent strings.

```
quotes_single = 'a string'
quotes_double = "a string"
```

- As shown in the code, we create two strings using single and double quotes, respectively.

- The strings created by using single and double quotes are the same.

Illegal characters in a string

- An example of an **illegal** character is a double quote inside a string that is surrounded by double quotes:

```
myString1 = "We are the so-called "Vikings" from the north."
myString2 = "I am 5'4""
```

- We will get an error if using double quotes inside a string that is surrounded by double quotes.

- To insert characters that are **illegal** in a string, **use an escape character**.

Escape characters and sequence

- The backslash character (\) is used to **“escape”** a special character in Python.

- The backslash character goes **in front of the character we want to “escape”**

```
print("I am 5'4\"")
#I am 5′4″
print ("We are the so-called \"Vikings\" from the north.")
#We are the so-called "Vikings" from the north.
```

- We call the combination of the backslash character and the character we want to “escape” is an **escape sequence**.

- Escape sequence are used to represent characters that are difficult or impossible to type directly in the code or that have a special meaning in Python

Common Escape Sequences (1/2)

Escape Sequence	Purpose
\'	Print a single quote
\"	Print a double quote
\	Print a backslash
\t	Print a tab
\n	Print a newline (“enter”)

Source: http://learnpythonthehardway.org/book/ex10.html

Common Escape Sequences (2/2)

```
str1 = "\tI'm tabbed in."
print(str1)
	I'm tabbed in.
```

\t adds a tab

```
str2 = "We are splitting\na line."
print(str2)
We are splitting
a line.
```

\n adds a newline

```
str3 = "I'm \\ a \\ good boy."
print(str3)
I'm \ a \ good boy.
```

\\ adds a single backslash

Source: http://learnpythonthehardway.org/book/ex10.html

Three ways to solve the problem of printing out inch using quotes

```
print("I am 5'4\"")
```

escape double quotes (using " for the entire string)

```
print('I am 5\'4"')
```

escape single quotes (using ' for the entire string)

```
print("I am 5\'4\"")
```

escape both single and double quotes (works for both ' and ")

Alternative solution: Triple Quotes

Triple quotes can **enclose** strings containing single and double quotes.

```
print('''She said, "Thank you! It's mine."''')
#She said, "Thank you! It's mine."


print('''I am 5'4"''')
#I am 5'4"
```

Triple Quotes can also build a multi-line string

- Another use case of the triple quotes is to represent a multi-line string, e.g.,

```
print('''Hello
World
!''')
```

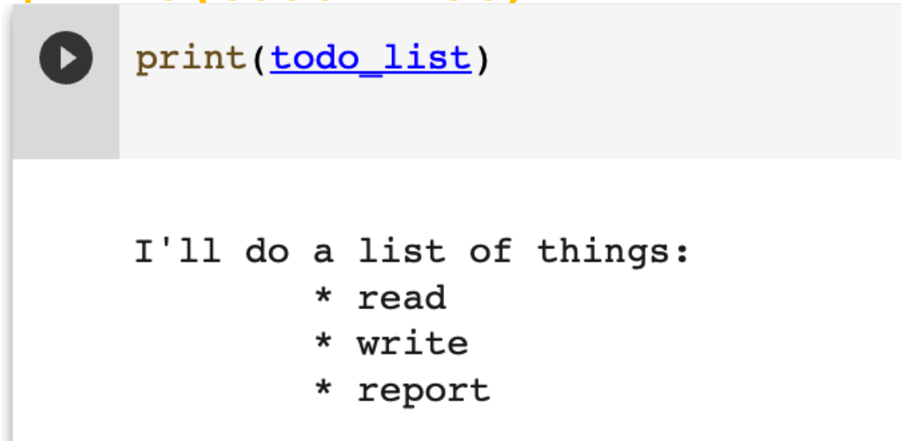
- We can also use escape sequences.

```
print('Hello\nWorld\n!')
```

Triple Quotes Example

```
todo_list= """
I'll do a list of things:
\t* read
\t* write
\t* report
"""
print(todo_list)
```

\t puts in a tab



```
print(todo_list)
```

```
I'll do a list of things:
        * read
        * write
        * report
```

When using triple quotes, the times you hit “**enter**” inside the string will print as newlines

Triple quotes are not an escape sequence but is helpful dealing with quotes in a string

Outline

- Arithmetic Operators
- Relational Operators
- Logical Operators
- String and Escape Characters
- **String Operations**

Types of string operations

- Concatenation/Addition:

```
str1 = "Simon Lee"
str2 = " is the best"
statement1 = str1 + str2
print(statement1)                #Simon Lee is the best
```

- Repetition/Multiplication:

```
“abc“ * 3   #arithmetic operator * on string

my_name = "Simon Lee "
new_name = 3 * my_name
print(new_name)
```

```
Simon Lee Simon Lee Simon Lee
```

Mixing numeric and string operands for string operations

```
x = 2
y = "hello"
print(x + y)  # What happens?

x = 2
y = "hello"
print(x * y) # What happens?
```

String Indexing

- You can access the characters in a string one at a time with the bracket operator, e.g., `letter = 'fruit'[1] #'r'`
  - extracts the character at index position **1** from the **fruit** variable and assigns it to the variable called **letter**.
  - The expression in brackets is called an ***index***, indicating which character in the sequence you want.

- Index starts from the position 0
  - In Python, the index starts from the beginning of the string, and the starting index is 0.

An example of string indexing (1/2)

```
Name = "Simon Lee"
```

String	S	i	m	o	n		L	e	e
Index	0	1	2	3	4	5	6	7	8

`Name[6]` What is the printout?

An example of string indexing (2/2)

```
Name = "Simon Lee"
```

String	S	i	m	o	n		L	e	e
Index	-9	-8	-7	-6	-5	-4	-3	-2	-1

`Name[-3]` What is the printout?

String indexing for escape sequences

- Escape sequences look like two characters to us
- But Python treats them as a **single** character

```
example1 = "see\n"
```

0	1	2	3
s	e	e	\n

```
example2 = "\tsee"
```

0	1	2	3
\t	s	e	e

String slicing

- A segment of a string is called a ***slice***. Selecting a slice is similar to selecting a character, but we are selecting multiple characters

```
s = 'Monty Python'
print(s[0:5])
print(s[6:12])
```

- The operator returns the part of the string from the "n-th" character to the "m-th" character, ***including*** the first but ***excluding*** the last.

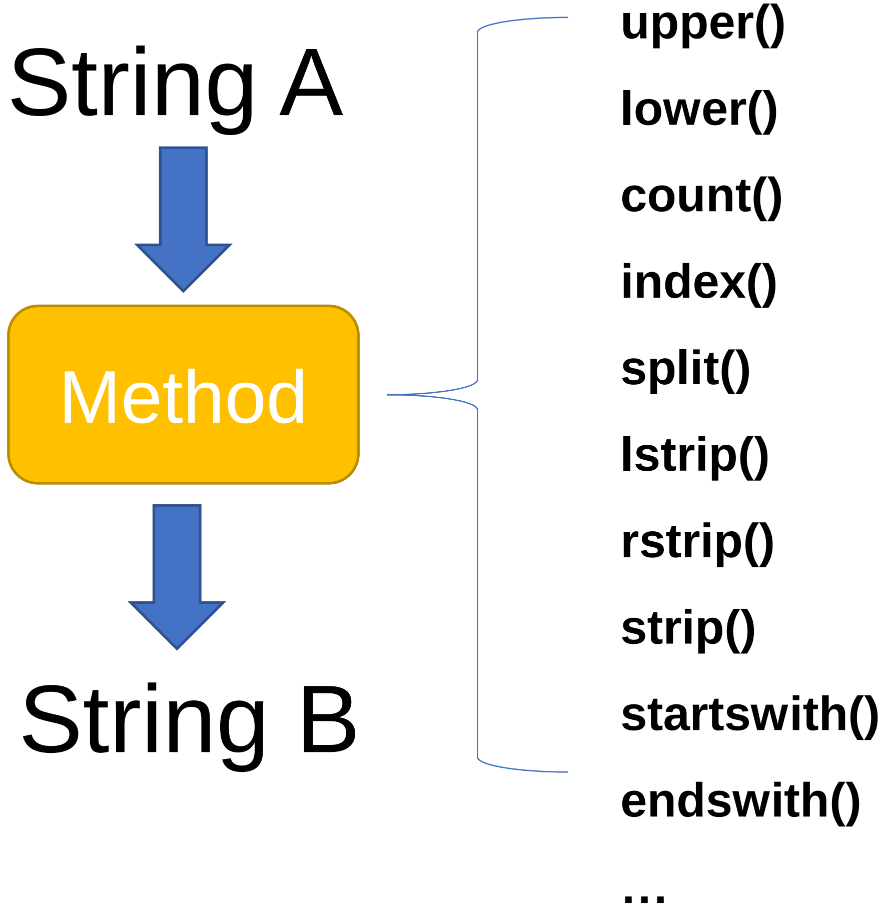
An example of string slicing

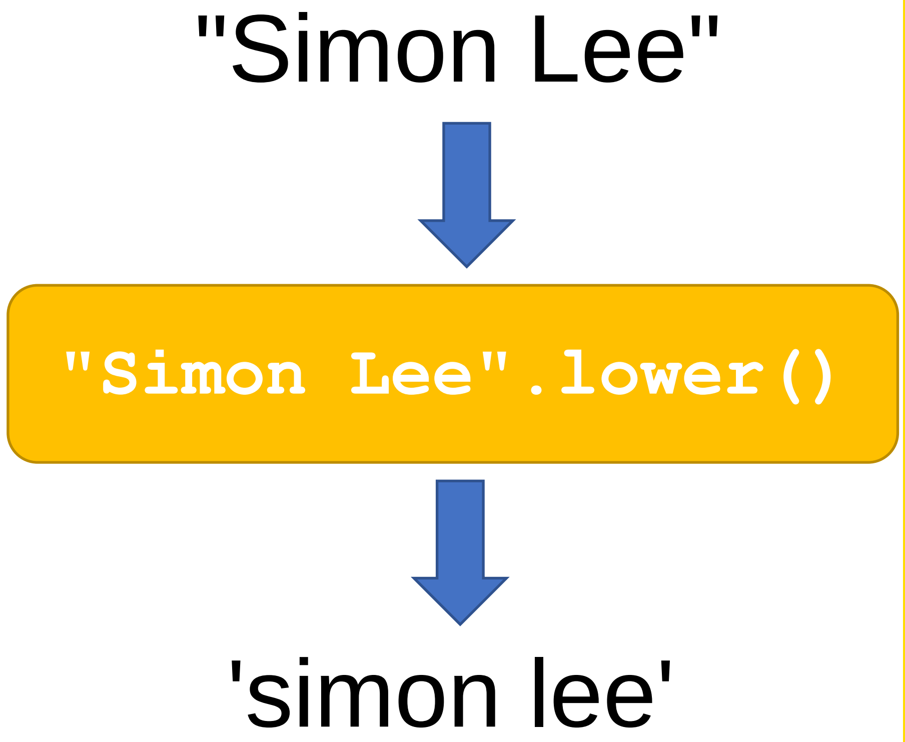
```
Name = "Simon Lee"
```

String	S	i	m	o	n		L	e	e
Index	0	1	2	3	4	5	6	7	8

```
firstName = Name[0:5]   # how to do this in Excel?
lastName = Name[6:9]    # how to do this in Excel?
```

String Methods (functions that belong to the string type)





Details on string methods (1/2)

- string.upper(): Converts the string into upper case
- string.lower(): Converts the string into lower case
- string.lstrip(): Returns a left trimmed version of the string
  - Space is the default leading character to remove
- string.rstrip(): Returns a right trimmed version of the string
  - Space is the default trailing character to remove
- string.strip(): Returns a trimmed version of the string
  - By default, remove any leading and/or trailing space character(s)

Details on string methods (2/2)

- string.count(): Returns occurrence of a specified value in the string
- string.index(): Searches the string for a specified value and returns the position of where it was found
- string.startswith(): Returns true if the string starts with the specified value
- string.endswith(): Returns true if the string ends with the specified value
- string.split(): Splits the string at the specified separator, and returns a list

String length

`len()` is a built-in function that returns the number of characters in a string

```
len("Simon Lee")

s = "Hello\nWorld" #\n is a newline character
len(s) # Returns 11. \n is two symbols, but it represents one
character
```

Is it a string method, i.e., does **`len()`** belong to string type?

String Operations

```
greet1 = "Hello World"
greet2 = "How are you?"

print(greet1.upper())
print(greet1.lower())
print(len(greet1))
print(greet1.index ("o"))
print(greet1.count('l'))
print(greet1[2:7])
print(greet1.startswith ("Hello"))
print(greet2.endswith ("you?"))

greetwords = greet1.split(" ")
print('greetwords[0] = ', greetwords[0])
print('greetwords[1] = ', greetwords[1])
```

What is the printout?

Recap Exercise

```
greet1 = "  Hello World   "
new_greet = greet1.replace("World", "John")
print(new_greet)
print(new_greet.lstrip())
print(new_greet.rstrip())
print(new_greet.strip())
```

What is the printout?

Recap Exercise

We already executed the following assignment statements:

```
width  = 20
height = 12.0
```

For each of the following expressions, write the value of the expression and the type (of the value of the expression)

```
width//2
width/2.0
height/3
1 + 2 ** 5
```

Use the Python interpreter to check your answers.

Recap Exercise

- Addition (+) for String means concatenation

```
print('a' + 'b’) # what is the printout?
```

- Multiplication (*) for String means Multiple concatenation

```
print('a' * 4) # what is the printout?
```