DSC510: Introduction to Data Science and Analytics
Lab 1: Introduction to Python

University of Cyprus
Department of Computer Science

Pavlos Antoniou

Office: B109, FST01

General Info

- Course website with material: https://piazza.com/ucy.ac.cy/fall2025/dsc510
- Submit lab assignments and project: https://moodle.cs.ucy.ac.cy

- Lab Instructor Information:
  - Email: antoniou.pavlos-AT-ucy.ac.cy
  - Office: B109 (Basement), Building ΘΕΕ01/FST01

- Lab-related assessment methods:
  - Lab assignments (10%): small exercises based on the lab material
  - Semester project (30%), done in groups of 3 students

Lab Schedule

- Lab01 (10/09): Setting up Python environment + Short Introduction to Python
- Lab02 (17/09): Data Manipulation (Pandas)
- Lab03 (24/09): Data Visualization (Matplotlib, Seaborn)
- Public Holiday 01/10
- Lab04 (08/10): Data Pre-processing 1: Cleaning, encoding, re-sampling, scaling
- Lab05 (15/10): Data Pre-processing 2: Feature selection/extraction, Dimensionality Reduction
- Lab06 (22/10): ML: Regression (linear, logistic, SVR, RFR)
- Lab07 (29/10): ML: Regression – cont’d
- Lab08 (05/11): ML: Classification (kNN, SVC, RFC)
- Lab09 (12/11): ML: Clustering (K-means)
- Lab10 (19/11): ML: Natural Language Processing
- Lab11 (26/11): ML: Timeseries
- Lab12 (03/12): No lab (project presentation week)

Using data to build models and make predictions

Recommended Lab Tools for writing code

- Install Anaconda locally and use **JupyterLab** (recommended) or Jupyter Notebook – work offline, use local resourses (CPU, RAM)

  or

- Google Colab – needs Google account and Internet access, use Google cloud resourses

Newlines and Whitespaces

- Use a newline to end a line of code.
  - Use \ when must go to next line prematurely.
- Whitespace is meaningful in Python: especially indentation
- No braces { } to mark blocks of code in Python…
  Use consistent indentation – whitespace(s) or tab(s) – instead.
  - The first line with more indentation starts a nested block
  - The first line with less indentation is outside of the block
  - Indentation levels must be equal within the same block but not necessarily the same with other blocks

```
if x%2 == 0:
   print("even")
   print("number")
else:
 print("odd")
```

- Often a colon appears at the start of a new block.
  - e.g. in the beginning of `if`, `else`, `for`, `while`, as well as of functions

Arithmetic Operators

Operator	Name	Examples
+	Addition	3 + 5 returns 8 "a" + "b" returns "ab"
-	Subtraction	50 - 24 returns 26
*	Multiplication	2 * 3 returns 6 "la" * 3 returns "lalala"
**	Exponentiation	3 ** 4 returns 81 (i.e. 3 * 3 * 3 * 3)
/	Division	4 / 3 returns 1.3333333333333333
//	Floor Division	4 // 3 returns 1 5.4 // 2.1 returns 2.0 (5.4/2.1 → 2.5714285714285716)
%	Modulus	8 % 3 returns 2 -25.5 % 2.25 returns 1.5

Enough to Understand the Code

- The basic printing command is `print()`
- Assignment uses `=` (e.g. `x=2`) and comparison uses `==`
- The first assignment to a variable creates it e.g.: `x = 8`
  - Variable types don’t need to be declared
  - Python figures out the variable types on its own
  - Multiple assignment is also available e.g.: `x, y = 2, 3`
- For numbers, arithmetic operators `+ - * / %` are as expected
  - Special use of `+` for string concatenation: `"Hello" + "World"`
  - Special use of `%` for string formatting (see the use of print() in extended Lab1)
- Logical operators are words (`and`, `or`, `not`) **not** symbols
  - e.g. `if x==2 and y>7:`

Comments

- Single line comments: Start comments with # – the rest of line is ignored by the python interpreter

```
# this is a single-line comment
```

- Multiple line comments: Start/end comments with """

```
"""
this is
a multi-line
comment
"""
```

Naming Rules

- Names (of variables or functions) are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob Bob _bob _2_bob_ bob_2 BoB
```

- There are some reserved words:

```
and, assert, break, class, continue, def, del,
elif, else, except, exec, finally, for, from,
global, if, import, in, is, lambda, not, or,
pass, print, raise, return, try, while
```

Data Types I: Numbers

- `int` (integers) → e.g. age, counts
- `float` (decimals) → e.g. price, temperature

- Examples
- `x = 10          # int`
- `y = 3.14        # float`

Data Types II: Strings

- Represent text (names, categories).

- Examples

```
name = "Alice"  # single quotes '' can be also used
print(f"Hello {name}")   # f-string
```

- In Python, we often need to **combine text with variables**. Instead of doing clunky concatenation like 'Hello ' + name, Python gives us a neat shortcut called f-strings.
- An **f-string is just a normal string but with an f in front**. Inside it, you can put variables or even expressions/variables in curly braces {} and Python will replace them with their values. This is super handy when printing results, debugging, or showing outputs.

Data Types III: Booleans

- Logical values: True, False.
- Important for filtering and conditions.

- Examples

```
• is_student = True    # set a variable to True
• print(5 > 3)         # prints the result of
                       # condition which is True
```

Data Types IV: Collections

- List → ordered, mutable (like a column of values).
- Tuple → ordered, immutable (like list but values cannot be changed).
- Set → unique elements (good for categories).
- Dict → key-value pairs (similar to JSON).

- Examples
- `nums = [1, 2, 3]      # list`
- `coords = (10, 20)     # tuple`
- `unique_vals = {1, 2, 3, "John"}           # set`
- `student = {"name": "Alice", "age": 23}    # dict`

Data access & Slicing

- We can access individual values of a tuple, list or string using square bracket “array” notation
  - Positive index: count from the left, starting with 0
  - Negative index: count from right, starting with –1
- `nums = [10, 20, 30, 40, 50]`
- `print(nums[0])          # first element`
- `print(nums[-1])         # last element`

- Slicing allows to retrieve a subset of the original collection using :
- `print(nums[1:4])        # [20, 30, 40]`
- `print(nums[:3])         # [10, 20, 30]`
- `print(nums[2:])         # [30, 40, 50]`

The ‘in’ operator for membership testing

- Boolean test whether a value is inside a collection:

```
nums = [1, 2, 4, 5]
print(3 in li)        # prints False
print(4 in li)        # prints True
print(4 not in li)    # prints False
```

- For strings, tests for substrings

```
a = "abcde"
print("c" in a)       # prints True
print("cd" in a)      # prints True
print("ac" in a)      # prints False
```

Conditional statements if/elif/else

- Used to check for condition(s)
- Conditions use comparison operators ==, !=, <, >, <=, >=
- Combine conditions with and, or, not
- Examples

```
age = 18
if age < 12:
   print("Child.")
elif age < 18:
   print("Teenager.")
else:
   print("Adult.")
print("This is outside the if statement.")
```

Loops I

- The `for` statement for predefined number of steps

```
for i in range(5):  # ranges from 0 to 4
   print(i)
print("Outside of the loop.")
```

Output:

```
0
1
2
3
4
Outside of the loop.
```

range()

- The range() function has two sets of parameters, as follows:
  - range(stop)
    - stop: Number of integers (whole numbers) to generate, starting from zero. E.g. `range(3)` → `[0, 1, 2]`
  - range([start], stop[, step])
    - start: Starting number of the sequence.
    - stop: Generate numbers up to, but not including this number.
    - step: Difference between each number in the sequence.

    E.g. `range(10,2,-2)` → `[10, 8, 6, 4]`
- Note that:
  - All parameters must be integers.
  - All parameters can be positive or negative.

Loops II

- The `for` statement for iterating over a list, string, tuple

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

Output:

```
apple
banana
cherry
```

```
text = "Hello"
for c in text:
   print(c)
```

Output:

```
H
e
l
l
o
```

Loops III

- The `while` statement

```
x = 0
while x < 5:
    print(x)
    x = x + 1
print("Outside of the loop.")
```

Output:

```
0
1
2
3
4
Outside of the loop.
```

User-defined functions

- def creates a function and assigns it a name
- return sends a result back to the caller

```
def <name>(arg1, arg2, ..., argN):
    <statements>
    return <values>
```

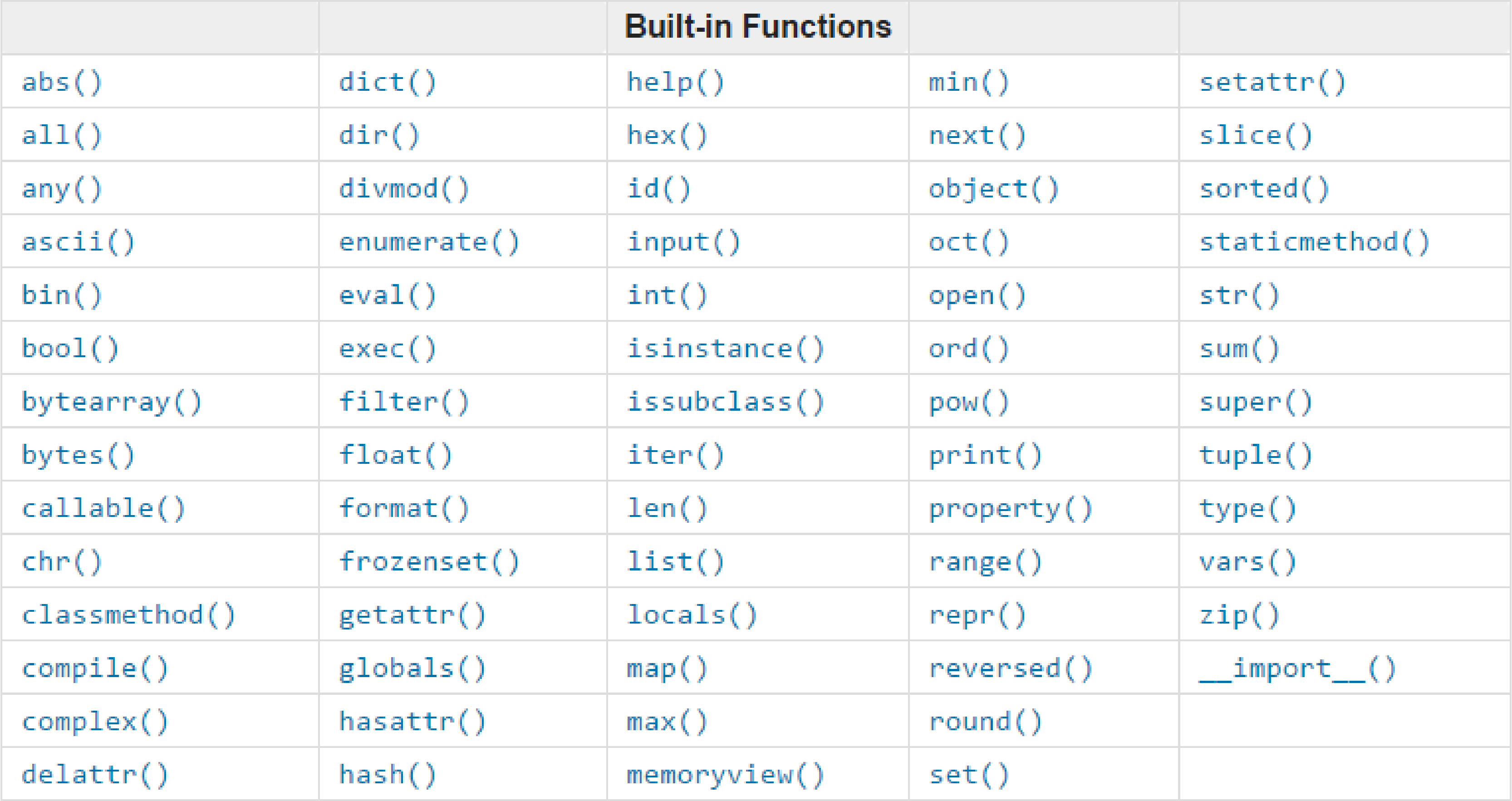
```
def times(x,y):
     return x*y
```

Function call:

```
x = times(4,5) # returns 20
```

Built-in functions

- https://docs.python.org/3/library/functions.html



		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

`len()` :

- Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

`min()` / `max()` :

- Return the smallest / largest item in an iterable or the smallest of two or more arguments.

Built-in functions: len(), max(), min()

```
my_list = ['one', 'two', 3]
my_list_len = len(my_list)  # length of my_list is 3
for i in range(0, my_list_len):
    print(my_list[i])
```

Output:

```
one
two
3
```

```
print(max("hello","world"))    # prints 'world'
print(max(3,13))               # prints 13
print(min([11,5,19,66]))       # prints 5
```

Modules

- Modules are functions and variables defined in separate files
- Items are imported using from or import

```
from module import function
function()
```

A’ Way

```
import module
module.function()
```

B’ Way

Mathematical functions

- https://docs.python.org/3.9/library/math.html

```
import math
print(math.sqrt(3))    # 1.7320508075688772

from math import sqrt
print(sqrt(3))         # 1.7320508075688772
```

Hands on

- Download Lab01_Python_Basics.ipynb from Piazza and follow the guidelines