# Introduction to Data Science Lab
# Fall 2025
# Lab 02 – Week 02

## Python Programming Cont.

### Data type

Python is a dynamically typed language, so we do not need to specify the type of a variable when we create one.

```
x=1.2
print(type(x))

#If we assign a new value to a variable, its type can change.
x=3
print(type(x))
```

```
<class 'float'>
<class 'int'>
```

```
#integers
x = 1
print (type(x))
```

```
<class 'int'>
```

```
# float
x = 1.0
print (type(x))
```

```
<class 'float'>
```

```
# boolean
b1 = True
b2 = False

print (type(b1))
```

```
<class 'bool'>
```

```
# complex numbers: note the use of `j` to specify the imaginary part
x = 1.0 - 1.0j
print (type(x))
```

```
<class 'complex'>
```

```
print(x)
print(x.real, x.imag)
```

```
(1-1j)
1.0 -1.0
```

Check if a certain variable is integer or float....

```
x= 5
print (type(x) is int)
```

```
True
```

```
x= 5.0
print (type(x) is int)
```

```
False
```

We can also use the `isinstance` method for testing types of variables:

```
print (isinstance(x, float))
```

```
True
```

## Type Casting

Typecasting in Python refers to the process of converting a variable from one data type to another. Python provides several built-in functions for typecasting. Here are some common typecasting functions:

```
# int(x)

float_num = 3.14
int_num = int(float_num)
print(int_num)
```

```
3
```

```
# float(x)

int_num = 5
float_num = float(int_num)
print(float_num)
```

```
5.0
```

```
# bool(x)

number = 0
bool_value = bool(number)
print(bool_value)
```

```
False
```

```
#list(x)

string = "hello"
list_chars = list(string)
print(list_chars)
```

```
['h', 'e', 'l', 'l', 'o']
```

Complex variables cannot be cast to floats or integers. We need to use `z.real` or `z.imag` to extract the part of the complex number we want:

```
z = 3 + 4.2j
float_z= float(z.real)
int_z  = int(z.imag)
print(float_z, int_z)
```

```
3.0 4
```

## Conditional Statements

Conditional statements allow a program to make decisions and execute different blocks of code based on conditions. In Python, conditions are usually written using comparison operators (==, !=, <, >, <=, >=) and logical operators (and, or, not).

```python
#if Statement

x = 10
if x > 5:
    print("x is greater than 5")
```

```
x is greater than 5
```

```python
#if-else Statement

x = 3
if x >= 5:
    print("x is greater than or equal to 5")
else:
    print("x is less than 5")
```

```
x is less than 5
```

```python
#if-elif-else Statement

marks = 72
if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
elif marks >= 60:
    print("Grade: C")
else:
    print("Grade: Fail")
```

```
Grade: C
```

```
#Nested if Statements

x = 20
if x > 10:
    if x % 2 == 0:
        print("x is greater than 10 and even")
    else:
        print("x is greater than 10 and odd")
```

```
x is greater than 10 and even
```

```
#Conditional Expressions (Ternary Operator)

marks = 18
status = "Pass" if marks >= 18 else "Fail"
print(status)
```

```
Pass
```

**Logical operators**

```
#and Operator

age = 20
salary = 30000

if age > 18 and salary > 25000:
    print("Eligible for loan")
```

```
Eligible for loan
```

```
#or Operator

day = "Sunday"
if day == "Saturday" or day == "Sunday":
    print("It's weekend!")
```

```
It's weekend!
```

```
#not Operator

is_raining = False
if not is_raining:
    print("You can go outside without an umbrella")
```

```
You can go outside without an umbrella
```

## Strings

A string is a sequence of characters enclosed in either single quotes ('), double quotes ("), or triple quotes ''' or """). Strings are widely used to store and manipulate text.

```
#1. Single Quotes (')

str1 = 'Hello'
#Strings enclosed in single quotes.
#Best when you want to include double quotes inside the string without escaping:
s = 'He said "Python is fun!"'
print(s)
```

```
He said "Python is fun!"
```

```
#2. Double Quotes (")
str2 = "Python"
#Strings enclosed in double quotes.
#Best when you want to include single quotes inside the string without escaping:
s = "It's a sunny day"
print(s)
```

```
It's a sunny day
```

```
#3. Triple Quotes (''' or """)
str3 = '''This is
a multi-line string'''
#Used for Multi-line strings
s = """This string
spans across
multiple lines"""
print(s)
```

```
This string
spans across
multiple lines
```

### 1. Accessing Strings

We can index a character in a string using `[]`:

Indexing starts with 0

We can extract a part of a string using the syntax `[start:stop]`, which extracts characters between index `start` and `stop` -1 (the character at index `stop` is not included):

```
s="Hello World"
print (s[0])
```

```
H
```

```
print (s[0:4])
```

```
Hell
```

```
print (s[4:5])
```

```
o
```

If we omit either (or both) of `start` or `stop` from `[start:stop]`, the default is the beginning and the end of the string, respectively:

```
print (s[:5])
```

```
Hello
```

```
print (s[6:])
```

```
World
```

```
print  (s[:])
```

```
Hello World
```

We can also define the step size using the syntax `[start:end:step]` (the default value for `step` is 1, as we saw above). This technique is called slicing.

```
print (s[::2])
print (s[:2:1])
```

```
HloWrd
He
```

## 2. String Operations

```
print ("str1", "str2", "str3")  # The print statement concatenates strings with a space
```

```
str1 str2 str3
```

```
print ("str1", 1.0, False, -1j)  # The print statements converts all arguments to strings
```

```
str1 1.0 False (-0-1j)
```

```
print ("str1" + "str2" + "str3") # strings added with + are concatenated without space
```

```
str1str2str3
```

```
#Repetition (*)
word = "Hi "
print(word * 3)
```

```
Hi Hi Hi
```

## 3. String Functions

Python provides many built-in functions to work with strings.

```
text = "  Python Programming  "

print(len(text))          # Length of string
print(text.lower())       # Convert to lowercase
print(text.upper())       # Convert to uppercase
print(text.strip())       # Remove spaces from start and end
print(text.replace("Python", "Java"))  # Replace words
print(text.split())       # Split into list
```

```
22
  python programming
  PYTHON PROGRAMMING
Python Programming
  Java Programming
['Python', 'Programming']
```

## 4. String Membership Operator

You can check if a substring exists inside a string.

```
sentence = "I love Python"
print("Python" in sentence)   # True
print("Java" not in sentence) # True
```

```
True
True
```

### 5. String Formatting

You can format strings using f-strings or format().

```
[ ]  name = "Ali"
     age = 23

     print(f"My name is {name} and I am {age} years old.")
     print("My name is {} and I am {} years old.".format(name, age))
```

```
My name is Ali and I am 23 years old.
My name is Ali and I am 23 years old.
```

## Zen of Python

The Zen of Python is a collection of guiding principles for writing computer programs in Python. It's like Python's philosophy, written as short, witty aphorisms by Tim Peters.

You can see it directly in Python by typing this in a Python interpreter:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

## Mathematical built-in functions

```python
print(abs(-5))           # 5
print(round(3.75, 1))    # 3.8
print(pow(2, 3))         # 8
print(divmod(7, 3))      # (2, 1)
print(min(3, 1, 2))      # 1
print(max(3, 1, 2))      # 3
print(sum([1,2,3]))      # 6
```

```
5
3.8
8
(2, 1)
1
3
6
```

# Lab Tasks

## Task # 01

Ask the user for: Name, Age and Favorite subject. Print the output using f-strings in this format: "My name is Ali, I am 20 years old, and my favorite subject is Python."

## Task # 02

Print:

- a + b
- "Hello " + c
- a * c (string repetition)

a=5, b=2.5, c= “Python”.

## Task # 03

Write a Python program that:
Defines variables x, y, and z. Prints results of:

- x // y
- int(x / y)
- x / y
- z + x
- z * y

Explains with comments why x // y and int(x / y) differ for negative numbers.
Uses isinstance() to check and print the type of each result.

## Task # 04

Write a program that:

- Takes two inputs x and y.
- If both are integers:
- Print whether x // y is equal to int(x / y).

- Print "Divisible" if y * (x // y) == x, otherwise "Not divisible".
- If any input is a float, cast both to int and print their sum.
- Otherwise, print "Invalid input".

## Task # 05

The Zen of Python says: “Beautiful is better than ugly.”
Write a program that: Takes an integer n.
Using if-elif-else only:

- If n is positive and divisible by 2 (without %), print "Beautiful".
- If n is positive but not divisible by 2, print "Not Beautiful".
- If n is negative, print "Ugly".

Add comments in your code explaining how these conditions reflect the Zen idea.

## Task # 06

Write a Python program that asks the user to enter **four numbers**.

- Find the largest and smallest of the four numbers.
- Check if all four numbers are equal.
- Check if they are in ascending order or descending order.
- Display all results with clear message

## Task # 07

Write a Python program that:

1. Creates two complex numbers a and b.
2. Performs addition, subtraction, multiplication, and division.
3. Prints the real & imaginary parts of a.
4. Prints the conjugate of b.
5. Prints the magnitude of a.