

# PYTHON PROGRAMMING

Assoc. Prof. Dr. Bora Canbula

Manisa Celal Bayar University  
Department of Computer Engineering



# Python Programming

**Instructor**

Assoc. Prof. Dr.  
Bora CANBULA

**Phone**

0 (236) 201 21 08

**Email**

bora.canbula@cbu.edu.tr

**Office Location**

Dept. of CENG

Office C233

**Office Hours**

4 pm – 5 pm, Mondays

**Course Overview**

Python Programming (Teams Code: gkdtw9f)

This course is about basic concepts in Python Programming. Students will learn the data structures, functions, classes, and the special structures of Python. They will also learn how to create basic desktop applications with a GUI, web applications as APIs, basic games as projects. In this course, we will use GitHub actions for assignments.

**Required Text**

Advanced Guide to Python 3 Programming, Springer, *John Hunt*

Python for Everybody, PY4E, *Charles Severance*

**Course Materials**

- Python 3.x (Anaconda is preferred)
- Jupyter Notebook from Anaconda
- Pycharm from JetBrains / Microsoft Visual Studio Code
- PC with a Linux distro or a Linux terminal in Windows 10/11.

**Course Schedule**

Week	Subject	Week	Subject
01	Basic Concepts in Python	08	Desktop Programming
02	Sequences	09	Desktop Programming
03	Functions	10	Web Programming
04	Decision Structures	11	Web Programming
05	Loops	12	Game Programming
06	Classes	13	Game Programming
07	Special Structures	14	Student Projects

# Python Programming

**Instructor**

Assoc. Prof. Dr.  
Bora CANBULA

**Phone**

0 (236) 201 21 08

**Email**

bora.canbula@cbu.edu.tr

**Office Location**

Dept. of CENG

Office C233

**Office Hours**

4 pm – 5 pm, Mondays

**Course Overview**

Python Programming (Teams Code: gkdtw9f)

This course is about basic concepts in Python Programming. Students will learn the data structures, functions, classes, and the special structures of Python. They will also learn how to create basic desktop applications with a GUI, web applications as APIs, basic games as projects. In this course, we will use GitHub actions for assignments.

**Required Text**

Advanced Guide to Python 3 Programming, Springer, *John Hunt*

Python for Everybody, PY4E, *Charles Severance*

**Course Materials**

- Python 3.x (Anaconda is preferred)
- Jupyter Notebook from Anaconda
- Pycharm from JetBrains / Microsoft Visual Studio Code
- PC with a Linux distro or a Linux terminal in Windows 10/11.

## Project Themes



Sports



Economy



Health

## Project Examples



Sports

- Predicting Game Outcomes
- Injury Risk Prediction
- Player Market Value Prediction
- Fan Engagement Analysis
- Athlete Performance Comparison



Economy

- Stock Market Prediction
- Cryptocurrency Price Prediction
- Consumer Spending Analysis
- Predicting Unemployment Rates
- Credit Scoring Model

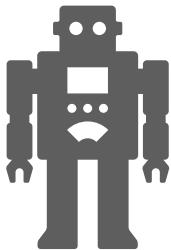


Health

- Disease Outbreak Prediction
- Personalized Health Recommendations
- Health Risk Prediction
- Hospital Readmission Prediction
- Nutritional Deficiency Prediction

## Natural Languages vs. Programming Languages

A language is a tool for expressing and recording thoughts.



Computers have their own language called **machine** language. Machine languages are created by humans, no computer is currently capable of creating a new language. A complete set of known commands is called an instruction list (IL).

The difference is that human languages developed naturally. They are still evolving, new words are created every day as old words disappear. These languages are called **natural** languages.



### Elements of a Language

- **Alphabet** is a set of symbols to build words of a certain language.
- **Lexis** is a set of words the language offers its users.
- **Syntax** is a set of rules used to determine if a certain string of words forms a valid sentence.
- **Semantics** is a set of rules determining if a certain phrase makes sense.



## Machine Language vs. High-Level Language

The IL is the alphabet of a machine language. It's the computer's mother tongue.

**High-level programming language** enables humans to write their programs and computers to execute the programs. It is much more complex than those offered by ILs.

A program written in a high-level programming language is called a **source code**. Similarly, the file containing the source code is called the **source file**.

## Compilation vs. Interpretation

There are two different ways of transforming a program from a high-level programming language into machine language:

**Compilation:** The source code is translated once by getting a file containing the machine code.

**Interpretation:** The source code is interpreted every time it is intended to be executed.

### Compilation

- The execution of the translated code is usually faster.
- Only the user has to have the compiler. The end user may use the code without it.
- The translated code is stored using machine language. Your code are likely to remain your secret.



### Interpretation

- You can run the code as soon as you complete it, there are no additional phases of translation.
- The code is stored using programming language, not machine language. You don't compile your code for each different architecture.



- The compilation itself may be a very time-consuming process
- You have to have as many compilers as hardware platforms you want your code to be run on.

- Don't expect interpretation to ramp up your code to high speed
- Both you and the end user have the interpreter to run your code.

## What is Python?

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

Python was created by Guido van Rossum. The name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.



Guido van Rossum

## Python Goals

- an **easy and intuitive** language just as powerful as those of the major competitors
- **open source**, so anyone can contribute to its development
- code that is as **understandable** as plain English
- **suitable for everyday tasks**, allowing for short development times



## Why Python?



- easy to learn
- easy to teach
- easy to use
- easy to understand
- easy to obtain, install and deploy

## Why not Python?



- low-level programming
- applications for mobile devices

## Python Implementations

An implementation refers to a program or environment, which provides support for the execution of programs written in the Python language.

- **CPython** is the traditional implementation of Python and it's most often called just "Python".
- **Cython** is a solution which translates Python code into "C" to make it run much faster than pure Python.
- **Jython** is an implementation follows only Python 2, not Python 3, written in Java.
- **PyPy** represents a Python environment written in Python-like language named RPython (Restricted Python), which is actually a subset of Python.
- **MicroPython** is an implementation of Python 3 that is optimized to run on microcontrollers.

## Start Coding with Python

- **Editor** will support you in writing the code. The Python 3 standard installation contains a very simple application named IDLE (Integrated Development and Learning Environment).
- **Console** is a terminal in which you can launch your code.
- **Debugger** is a tool, which launches your code step-by-step to allow you to inspect it.

first.py

```
print("Python is the best!")
```



<https://forms.office.com/r/z2XZbbypBR>

## Function Name

A function can cause some effect or evaluate a value, or both.

## Where do functions come from?

- From Python itself
- From modules
- From your code

**first.py**

```
print("Python is the best!")
```

## Argument

- Positional arguments
- Keyword arguments

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Output buffering is usually determined by *file*. However, if *flush* is true, the stream is forcibly flushed.

## Literals

A literal is data whose values are determined by the literal itself. Literals are used to encode data and put them into code.

### **literals.py**

```
print("7")
print(7)
print(7.0)
print(7j)
print(True)
print(0b10)
print(0o10)
print(0x10)
print(7.4e3)
```

- String
- Integer
- Float
- Complex
- Boolean
- Binary
- Octal
- Hexadecimal
- Scientific Notation

## Basic Operators

An operator is a symbol of the programming language, which is able to operate on the values.

### Multiplication

```
print(2 * 3) Integer
```

```
print(2 * 3.0) Float
```

```
print(2.0 * 3) Float
```

```
print(2.0 * 3.0) Float
```

### Division

```
print(6 / 3) Float
```

```
print(6 / 3.0) Float
```

```
print(6.0 / 3) Float
```

```
print(6.0 / 3.0) Float
```

### Exponentiation

```
print(2**3) Integer
```

```
print(2**3.0) Float
```

```
print(2.0**3) Float
```

```
print(2.0**3.0) Float
```

### Floor Division

```
print(6 // 3) Integer
```

```
print(6 // 3.0) Float
```

```
print(6.0 // 3) Float
```

```
print(6.0 // 3.0) Float
```

### Modulo

```
print(6 % 3) Integer
```

```
print(6 % 3.0) Float
```

```
print(6.0 % 3) Float
```

```
print(6.0 % 3.0) Float
```

### Addition

```
print(-8 + 4) Integer
```

```
print(-4.0 + 8) Float
```

## Operator Priorities

An operator is a symbol of the programming language, which is able to operate on the values.

### priorities.py

```
print(9 % 6 % 2)
print(2**2**3)
print(2 * 3 % 5)
print(-3 * 2)
print(-2 * 3)
print(-(2 * 3))
```

- + (unary)
- - (unary)
- \*\* (right-sided binding)
- \*
- /
- //
- % (left-sided binding)
- + (binary)
- - (binary)

## Variables

Variables are symbols for memory addresses.

# Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

### Built-in Functions

**A**

- `abs()`
- `aiter()`
- `all()`
- `anext()`
- `any()`
- `ascii()`

**E**

- `enumerate()`
- `eval()`
- `exec()`

**F**

- `filter()`
- `...`

**L**

- `len()`
- `list()`
- `locals()`

**M**

- `map()`
- `...`

**R**

- `range()`
- `repr()`
- `reversed()`
- `round()`

**S**

- `...`

### hex(x)

Convert an integer number to a lowercase hexadecimal string prefixed with “0x”. If `x` is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

`classmethod()`  
`compile()`  
`complex()`

`help()`  
**hex()**

`ord()`

`type()`

**D**

`id()`

`P`  
`pow()`  
`print()`

`V`  
`vars()`

### id(object)

Return the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

## Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.



### keyword — Testing for Python keywords

Source code: [Lib/keyword.py](#)

This module allows a Python program to determine if a string is a **keyword** or **soft keyword**.

**keyword.iskeyword(s)**

Return `True` if *s* is a Python **keyword**.

**keyword.kwlist**

Sequence containing all the **keywords** defined for the interpreter. If any keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

**keyword.issoftkeyword(s)**

Return `True` if *s* is a Python **soft keyword**.

*New in version 3.9.*

**keyword.softkwlist**

Sequence containing all the **soft keywords** defined for the interpreter. If any soft keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

*New in version 3.9.*



<https://forms.office.com/r/y6xQaXSL1F>

## Your First Homework



### Week01/info\_firstname\_lastname.py

- A string variable with the name student\_id that contains your student id.
- A string variable with the name full\_name that contains your full name.



### Week02/types\_firstname\_lastname.py

- An integer with the name: my\_int
- A float with with the name: my\_float
- A boolean with the name: my\_bool
- A complex with the name: my\_complex



### Rules for your pull requests

- Please run your code first in your computer, do not submit codes with syntax errors.
- Submit your code to WeekXX folder. WeekXX/hw is for me to move accepted works.
- If a change is requested, please edit the existing pull request, don't open a new one.

## Your First Homework



### Week01/info\_firstname\_lastname.py

- A string variable with the name student\_id that contains your student id.
- A string variable with the name full\_name that contains your full name.



### Week02/types\_firstname\_lastname.py

- An integer with the name: my\_int
- A float with with the name: my\_float
- A boolean with the name: my\_bool
- A complex with the name: my\_complex

The screenshot shows a GitHub profile for the user 'canbula'. The profile picture is a circular photo of a man with a bald head, wearing a dark t-shirt with a yellow Python logo. The profile name is 'Bora Canbula' and the handle is 'canbula'. Below the profile, there is a button for 'Unfollow' and a status message: 'Assoc. Prof. Dr. @mcbuceng studying'. The user has 27 repositories, 50 stars, and 12 followers. The 'Pinned' section displays five repositories:

- PythonProgramming** (Public) - Repository for Python Programming course given by Assoc. Prof. Dr. Bora Canbula at Computer Engineering Department of Manisa Celal Bayar University. (Jupyter Notebook, 11 stars, 29 forks)
- Statistics** (Public) - Repository for Statistics course given by Assoc. Prof. Dr. Bora Canbula at Department of Computer Engineering, Manisa Celal Bayar University. (Python, 21 stars, 12 forks)
- ParallelProgramming** (Public) - Repository for Parallel Programming course given by Assoc. Prof. Dr. Bora Canbula at Computer Engineering Department of Manisa Celal Bayar University. (Python, 55 stars, 117 forks)
- NumericalAnalysis** (Public) - Repository for Numerical Analysis course given by Assoc. Prof. Dr. Bora Canbula at Computer Engineering Department of Manisa Celal Bayar University. (Jupyter Notebook, 25 stars, 68 forks)
- MinimalFletClass** (Public template) - A minimal class which generates a Flet application. (Python, 8 stars, 1 fork)
- ieee754** (Public) - Python module which finds the IEEE-754 representation of a floating point number. (Python, 22 stars, 3 forks)

# Equality & Identity & Comparison

## Equality

The operators `is` and `is not` test for an object's identity: `x is y` is true if and only if `x` and `y` are the same object. An Object's identity is determined using the `id()` function. `x is not y` yields the inverse truth value. [4]

```
a = 4
print(a == 4) ✓
print(id(a) == id(4)) ✓
print(a is 4) ✓
print(id(a) is id(4)) ✗
```

## Left- or Right-sided?

```
x, y, z = 0, 1, 2
print(x == y == z) ✗
print(x == (y == z)) ✓
print((x == y) == z) ✗
```

## Inequality

```
print(1 != 2) ✓
print(not 1 == 2) ✓
```

## Updated Priority Table

Operator	Type
<code>+, -</code>	unary
<code>**</code>	binary
<code>*, /, //, %</code>	binary
<code>+, -</code>	binary
<code>&lt;, &lt;=, &gt;, &gt;=</code>	binary
<code>!=, ==</code>	binary

## Comparison

```
print(1 < 2) ✓
print(1 <= 2) ✓
print(1 > 2) ✗
print(1 >= 2) ✗
```

## Chaining

```
print(1 < 2 < 3) ✓
print(1 < 2 > 3) ✗
print(1 < 2 >= 3) ✗
print(1 < 2 <= 3) ✓
```

QUESTION

Using one of the comparison operators in Python, write a simple two-line program that takes the parameter `n` as input, which is an integer, prints `False` if `n` is less than 100, and `True` if `n` is greater than or equal to 100.

```
n = int(input())
print(n >= 100)
```

# Conditional Execution

## if statement

```
if n >= 100:
    print("The number is greater than or equal to 100.")
elif n < 0:
    print("The number is negative.")
else:
    print("The number is less than 100.")
```

## Ternary Operator

```
msg = "The number is greater than or equal to 100." if n >= 100 else "The number is less than 100."
print(msg)
```

# Loops

## QUESTION

- The program generates a random number between 1 and 10.
- The user is asked to guess the number.
- The user is given feedback if the guess is too low or too high.
- The user is asked to guess again until the correct number is guessed.

```
r = random.randint(1, 10)
answer = False
while not answer:
    n = int(input("Enter a number: "))
    if n == r:
        print("You guessed it right!")
        answer = True
    elif n < r:
        print("Try a higher number.")
    else:
        print("Try a lower number.")
```

## QUESTION

- The user is asked to enter a number.
- The program prints the numbers from 0 to n-1.

```
n = int(input("Enter a number: "))
for i in range(n):
    print(i, end=" ")
```

## break and continue

```
n = int(input("Enter a number: "))
for i in range(10):
    if i < n:
        print("The number is not found:", i)
        continue
    if i == n:
        print("The number is found:", i)
        break
```



```
r = random.randint(1, 10)
while True:
    n = int(input("Enter a number: "))
    if n == r:
        print("You guessed it right!")
        break
    elif n < r:
        print("Try a higher number.")
    else:
        print("Try a lower number.")
```

### class range(stop)

### class range(start, stop[, step])

The arguments to the range constructor must be integers (either built-in `int` or any object that implements the `__index__()` special method). If the `step` argument is omitted, it defaults to 1. If the `start` argument is omitted, it defaults to 0. If `step` is zero, `ValueError` is raised.

For a positive `step`, the contents of a range `r` are determined by the formula `r[i] = start + step*i` where `i >= 0` and `r[i] < stop`.

For a negative `step`, the contents of the range are still determined by the formula `r[i] = start + step*i`, but the constraints are `i >= 0` and `r[i] > stop`.

#### start

The value of the `start` parameter (or 0 if the parameter was not supplied)

#### stop

The value of the `stop` parameter

#### step

The value of the `step` parameter (or 1 if the parameter was not supplied)

The advantage of the `range` type over a regular `list` or `tuple` is that a `range` object will always take the same (small) amount of memory, no matter the size of the range it represents (as it only stores the `start`, `stop` and `step` values, calculating individual items and subranges as needed).

**Can we use `while/for` with `else`?**