



The Islamic University of Gaza
Faculty of Engineering
Computer Engineering Department



Introduction to Computers

LAB #2

Elementary Programming I

Eng. Hashem M. Hejazy

Eng. Mohammed O. AlHabbash

Spring 2026

Variables

In the early days to store or retrieve something you should know the memory location of that data. In modern programming languages, we use **variables**. **Variables** are the names that reference values stored in memory. They are called **variables** because they may reference different values. The statement for assigning a value to a variable is called an **assignment statement**. In Python, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is `variable = expression`. Here are some examples of assignment statements:

```
x = 1
height = 5.5
y = 4 * (5 - 3) + 2 / 4
z = x + y
squareArea = height * height
```

As we see, an expression represents a computation involving values, variables, and operators that evaluate to a value. You can use a variable in an expression. The same variable can also be used on both sides of the = operator. For example,

```
x = 1
x = x + 1
```

In this assignment statement, the result of $x + 1$ is assigned to x . If x is 1 before the statement is executed, then it becomes 2 after the statement is executed.

Note: In mathematics, $x = 2 * x + 1$ denotes an equation. However, in Python, $x = 2 * x + 1$ is an assignment statement that evaluates the expression $2 * x + 1$ and assigns the result to x .

To assign a value to a variable, you must place the variable name to the left of the assignment operator. Thus, the following statement is wrong:

```
1 = x # Wrong
```

If a value is assigned to multiple variables, you can use a syntax called multiple assignments:

```
i = j = k = 1
```

which is equivalent to

```
k = 1
j = k
i = j
```

A variable must be declared and assigned a value before it can be used. For example, the following code is wrong:

```
j = i + 1
```

If we tried to execute the previous example we will get the following error:

```
NameError: name 'i' is not defined.
```

To fix it, we may write the code like this:

```
i = 0
j = i + 1
```

Comments

Comments are text inside of your source code that is ignored by the interpreter. Comments help the people to read the code, which is you most of the time, better understand the intent, structure, and functionality of the program. Also, comments are frequently used to hide parts of the source code without completely deleting them. There are two types of comments in Python:

1. Line Comments `#`, the interpreter ignores everything until the end of the line.
2. Block Comments `""" ... """`, also called paragraph comments, the interpreter ignores everything between those notations. When it sees `"""`, it scans for the next `"""` and ignores any text between the triple quotation marks. They can happen in one or multiple lines.

```
'''
This is a
    Block Comment
'''
# This is a Line Comment
```

Simple Program

Let's say we are required to write a program that computes the area of a circle. To write programs, we actually do two things:

1. Designing algorithms.
2. Translating algorithms into programming instructions or code.

An **algorithm** describes how a problem is solved by listing the actions that need to be taken and the order of their execution. The algorithm for calculating the area of a circle is:

- Get the radius
- Compute the area using this formula: $A = \pi * r^2$
- Print the result.

In the first step, the radius must be stored in the program. So, we use variables to store the radius value and access it. Next, we compute the area using the formula and store the result in another variable. Finally, we print the area on the screen using the `print` function. Here is the complete program:

```
# Declare the radius variable and assign a value to it
radius = 20
# Compute area using the formula
area = 3.14159 * radius * radius
# Print the result
print("Area of the circle with radius", radius, "is", area)
```

Identifiers

Identifiers are the names that identify the elements such as variables and functions in a program. As we saw earlier, `radius`, `circumference`, `height`, and `print` are names of variables or functions. These names are called identifiers. All identifiers in a program must obey the following rules:

- An identifier is a sequence of characters that consists of letters (upper or lower case), digits, and underscores (_).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- Keywords (Reserved Words) cannot be used as an identifier.

The following table contains all python keywords, which you cannot use as identifiers for variables or functions.

<code>and</code>	<code>else</code>	<code>in</code>	<code>return</code>	<code>as</code>
<code>except</code>	<code>is</code>	<code>True</code>	<code>assert</code>	<code>False</code>
<code>lambda</code>	<code>try</code>	<code>break</code>	<code>finally</code>	<code>None</code>
<code>while</code>	<code>class</code>	<code>for</code>	<code>nonlocal</code>	<code>with</code>
<code>continue</code>	<code>from</code>	<code>not</code>	<code>yield</code>	<code>def</code>
<code>global</code>	<code>or</code>	<code>del</code>	<code>if</code>	<code>pass</code>
<code>elif</code>	<code>import</code>	<code>raise</code>		

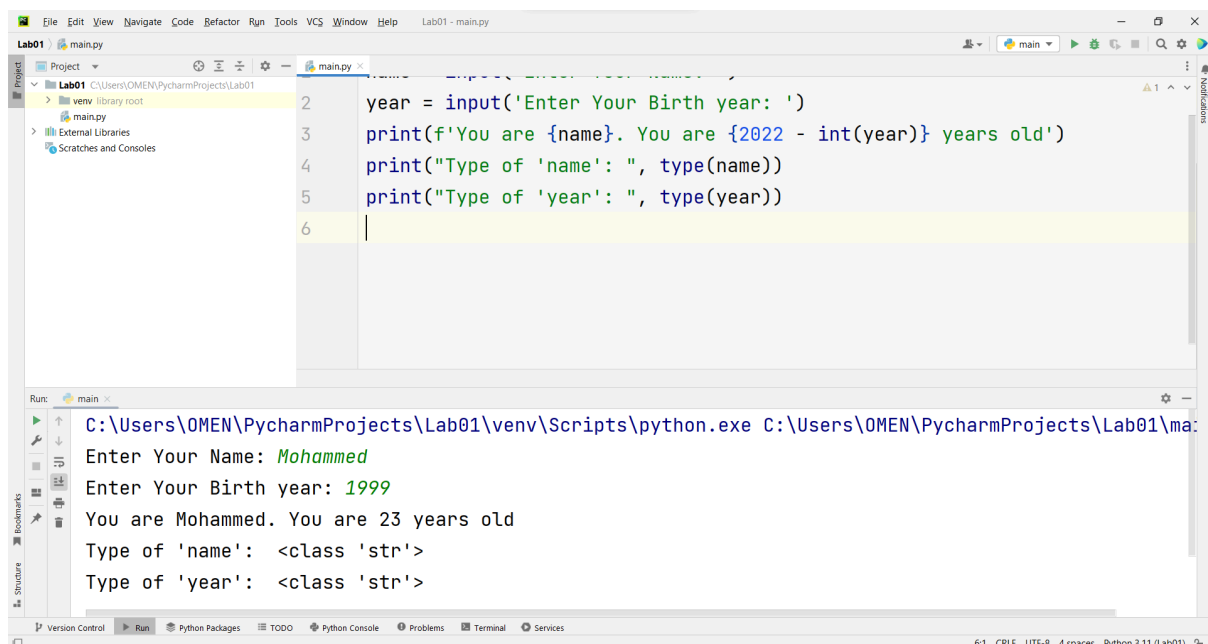
Some notes about identifiers:

- Python is **case sensitive**. So, `height`, `Height`, and `HEIGHT` are different identifiers.
- You **cannot use spaces** in identifiers, so if a name consists of several words, concatenate them into one, connecting them with underscores (_). This style is called **snake_case**. Another one is called **camelCase**, in which you connect words by making the first word in lowercase and capitalizing the first letter of each subsequent word. However, in **Python**, it's **recommended to code with snake_case**.

Reading User Input

In the previous example, the radius is set in the source code. If we want to change the radius we need to modify the source code, which is impractical. We can use the **input** function to read values from the user. The **input** function prompts the user to enter a value and returns the value which will be stored in a variable. The following example illustrates the **input** function:

```
name = input('Enter Your Name: ')
year = input('Enter Your Birth year: ')
print(f'You are {name}. You are {2023 - int(year)} years old')
print("Type of 'name': ", type(name))
print("Type of 'year': ", type(year))
```



*Note that: the **input** function return always a string value, and to use its numeric value you need to **cast** it. We will talk about casting in the coming labs.*

Previously, we used comma (,) to print multiple values in the print function. Python allows us to write expressions inside the string literal using **f-strings** (f'...{expression}..'). This lets you write less code, and makes the code easier to read.

Now, we need to modify the code that computes a circle's area to read the radius from the user, not hard coded.

```
radius = float(input('Enter radius: '))
# Compute area using the formula
area = 3.14159 * radius * radius
# Print the result
print(f'Area of the circle with radius {radius} is {area}')
```

Note that the radius can be a floating-point value, so we need to cast it to the datatype **float**. We'll be talking about datatypes in later labs.

Simultaneous Assignment

In Java if we want to swap two variables, we need a lot of code. Suppose we have two variables x and y. x value is 1 and y value is 2. Here is the code to swap the two variables in Java:

```
x = 1, y = 2;
temp = x;
x = y;
y = temp;
```

In Python, we can swap the two variables with just one statement instead of three in Java:

```
x = 1
y = 2
x, y = y, x
```

The last statement is called **simultaneous assignment**:

```
variable1, variable2 = expression1, expression2
```

Of course, in the last code we could use simultaneous assignment to assign x and y like this:

```
x, y = 1, 2
x, y = y, x
```

Constants

A constant is an identifier that represents a permanent value. The value of a variable may change during the execution of a program, but constants represent permanent data that never changes. For example, in the circle's area program, π is a constant and it never changes, so instead of typing **3.14159**, we can use a descriptive name (constant) **PI** for the value.

Python does not have a special syntax for constants. We use variables to denote constants, but **to distinguish a constant from a variable, use all uppercase letters to name a constant.**

```
radius = float(input('Enter radius: '))
PI = 3.14159
area = PI * radius * radius
print(f'Area of the circle with radius {radius} is {area}')
```

Benefits of using constants:

- If the value is used in different locations, you only type it once.
- If you have to change the value, you only change one location.
- Descriptive identifiers make the program easy to read.

Lab tasks

1. Write a program that calculates simple interest. The program asks the user for the principal amount, annual interest rate, and number of years, then calculates the simple interest using:

$$\text{Interest} = \text{Principal} \times \text{Rate} \times \text{Years}.$$

2. Write a program that converts USD to ILS (Israeli Shekels). The program prompts the user to enter an amount in USD, converts it to ILS using a fixed exchange rate (e.g., 1 USD = 3.25 ILS), and then displays the result.

Good Luck 😊