Module 2: Data types, variables, basic input-output operations, basic operators

2.4.1 Variables

Python Variable is containers that store values. Python is not “statically typed”. We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it.

A Python variable is a name given to a memory location. It is the basic unit of storage in a program

If you want to **give a name to a variable**, you must follow some strict rules:

- the name of the variable must be composed of upper-case or lower-case letters, digits, and the character _ (underscore)
- the name of the variable must begin with a letter;
- the underscore character is a letter;
- upper- and lower-case letters are treated as different (a little differently than in the real world - *Alice* and *ALICE* are the same first names, but in Python they are two different variable names, and consequently, two different variables);
- the name of the variable must not be any of Python's reserved words (the keywords - we'll explain more about this soon).

2.4.2 Keywords

Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name or any other identifier.

Value Keywords: True, False, None

There are three Python keywords that are used as values. These values are singleton values that can be used over and over again and always reference the exact same object. ou’ll most likely see and use these values a lot.

Operator Keywords: and, or, not, in, is

Several Python keywords are used as operators. In other programming languages, these operators use symbols like &, |, and !. The Python operators for these are all keywords:

Math Operator	**Other Languages**	**Python Keyword**
AND, ∧	&&	and
OR, ∨	||	or
NOT, ¬	!	not
CONTAINS, ∈		in
IDENTITY	===	is

Control Flow Keywords: if, elif, else

Three Python keywords are used for control flow: if, elif, and else. These Python keywords allow you to use conditional logic and execute code given certain conditions.

Iteration Keywords: for, while, break, continue, else

Looping and iteration are hugely important programming concepts. Several Python keywords are used to create and work with loops. These, like the Python keywords used for conditionals above, will be used and seen in just about every Python program you come across. Understanding them and their proper usage will help you improve as a Python programmer.

Structure Keywords: def, class, with, as, pass, lambda

In order to define functions and classes or use context managers, you'll need to use one of the Python keywords in this section. They're an essential part of the Python language, and understanding when to use them will help you become a better Python programmer.

Returning Keywords: return, yield

There are two Python keywords used to specify what gets returned from functions or methods: return and yield. Understanding when and where to use return is vital to becoming a better Python programmer. The yield keyword is a more advanced feature of Python, but it can also be a useful tool to understand.

Import Keywords: import, from, as

For those tools that, unlike Python keywords and built-ins, are not already available to your Python program, you'll need to import them into your program. There are many useful modules available in Python's standard library that are only an import away.

Exception-Handling Keywords: try, except, raise, finally, else, assert

One of the most common aspects of any Python program is the raising and catching of exceptions. Because this is such a fundamental aspect of all Python code, there are several Python keywords available to help make this part of code clear and concise.

Asynchronous Programming Keywords: async, await

Asynchronous programming is a complex topic. There are two Python keywords defined to help make asynchronous code more readable and cleaner: async and await.

Variable Handling Keywords: del, global, nonlocal

Three Python keywords are used to work with variables. The del keyword is much more commonly used than the global and nonlocal keywords.

2.4.3 Creating Variables

What can you put inside a variable?

Anything.

You can use a variable to store any value of any of the already presented kinds, and many more of the ones we haven't shown you yet.

The value of a variable is what you have put into it. It can vary as often as you need or want. It can be an integer one moment, and a float a moment later, eventually becoming a string.

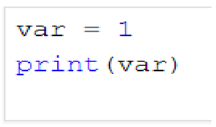
Let's talk now about two important things - **how variables are created**, and **how to put values inside them** (or rather - how to give or **pass values** to them).

A variable comes into existence as a result of assigning a value to it. Unlike in other languages, you don't need to declare it in any special way.

If you assign any value to a nonexistent variable, the variable will be **automatically created**. You don't need to do anything else.

The creation (or otherwise - its syntax) is extremely simple: **just use the name of the desired variable, then the equal sign (=) and the value you want to put into the variable.**

Take a look at the snippet:



```
var = 1
print(var)
```

It consists of two simple instructions:

- The first of them creates a variable named var, and assigns a literal with an integer value equal to 1.
- The second prints the value of the newly created variable to the console.

Assigning a new value to an already existing variable

How do you assign a new value to an already created variable? In the same way. You just need to use the equal sign.

The equal sign is in fact an assignment operator. Although this may sound strange, the operator has a simple syntax and unambiguous interpretation.

It assigns the value of its right argument to the left, while the right argument may be an arbitrarily complex expression involving literals, operators and already defined variables.

Look at the code below:

```
var = 1

print(var)

var = var + 1

print(var)
```

The code sends two lines to the console:

```
1

2
```

The first line of the snippet creates a new variable named var and assigns 1 to it.

The statement reads: assign a value of 1 to a variable named var.

We can say it shorter: assign 1 to var.

Some prefer to read such a statement as: var becomes 1.

The third line assigns the same variable with the new value taken from the variable itself, summed with 1. Seeing a record like that, a mathematician would probably protest - no value may be equal to itself plus one. This is a contradiction. But Python treats the sign = not as equal to, but as assign a value.

So how do you read such a record in the program?

Take the current value of the variable var, add 1 to it and store the result in the variable var.

In effect, the value of variable var has been incremented by one, which has nothing to do with comparing the variable with any value.

2.4.4 Shortcut Operators

It's time for the next set of operators that make a developer's life easier.

Very often, we want to use one and the same variable both to the **right and left sides of the = operator**.

For example, if we need to calculate a series of successive values of powers of 2, we may use a piece like this:

x = x * 2

You may use an expression like this if you can't fall asleep and you're trying to deal with it using some good, old-fashioned methods:

sheep = sheep + 1

Python offers you a shortened way of writing operations like these, which can be coded as follows:

x *= 2

sheep += 1

Let's try to present a general description for these operations.

If **op** is a two-argument operator (this is a very important condition) and the operator is used in the following context:

variable = variable op expression

It can be simplified and shown as follows:

variable op= expression

Take a look at the examples below. Make sure you understand them all.

i = i + 2 * j ⇒ i += 2 * j

var = var / 2 ⇒ var /= 2

rem = rem % 10 ⇒ rem %= 10

j = j - (i + var + rem) ⇒ j -= (i + var + rem)

x = x ** 2 ⇒ x **= 2