Module 1. Variables, Datatypes, and Operations

Functions

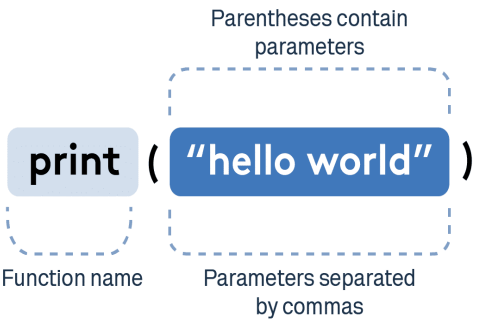
A function: a reusable block of code that can be executed many times within the program. When you need to use it, you just call it.

By using the print statement, we can see the output of our code and better understand how our program is working.

This can be done by calling **the print() function**.

To call a function, all you need to do is to type the function's name followed by **()** and place a desired argument in parentheses.

(Look at the example on the right)



For example:

```
print = (‘Hello World’)
```

Variables

Variables provide us with a way to store and manipulate data within a program. You can think of a variable as **a container.**

The **equal sign (=)** is used to assign values to variables.

The element on the left of the equal sign is the **variable name**, while the element on the right is the **value** stored in the variable.

For example:

```
name = 'Tom'
number = '4'
```

Variable names are **CASE-SENSITIVE.**

If we give the same name to two variables, but they differ in their capitalization, they are considered different variables (for example: *my_variable* and *My_Variable*).

A value has a **datatype**.

There are two main numeric categories:

1. **Integers:** whole numbers like **3** and **182;** or

2. **Floats:** decimal numbers like **3.14** and **182.03**

The other important datatype is **the string**.

An important difference in the syntax is that **strings should be written between quotation marks, either single or double.**

The number one rule for naming a variable:

The name can contain both lower - and uppercase letters (*but it does matter which one you use!*), digits (*but it can not start with a digit!*), and the underscore character (_).

✓ Things you might want to do:	✗ It's not the best idea to:
• Communicate what the variable contains.	• Use single-letter variables (there are one or two exceptions to this)
• If there's more than one "thing" in the variable, use the plural form (players instead of player)	• Use unclear abbreviations (std_grd_avg use student_grade_average instead)
• Use as many words as necessary (and not more)	• Put the type of the variable in the name (so no

	int_number_of_students)
• Change a variable name later on when you think something better.	• Use more words than necessary (the_average_grade_of_this_student vs student_grade_average).

Naming Conventions

The code needs to be readable and maintainable.

Key points:

Point 1: The use of the Snake_case	Use **snake case** for variable and function names. Snake case means lowercase variables where space would-be are replaced by *underscores* (**_**). • **winc_academy** • **answer_options**
Point 2: The use of space	Separate variables and operators with spaces, except in function calls. • **answer = 2 * 21** • **answer = get_answer(question='What time is it?')**
Point 3: Use descriptive names	• Choose meaningful and descriptive names for variables, functions, and classes. • This improves code readability and makes it easier for others (and yourself) to understand the purpose of each element.
Point 4: Capitalized Words	• **For naming classes**, it is recommended to use

	CapitalizedWords or PascalCase convention. • This means starting each word with a capital letter without any underscores. For example: **MyClass, MyClassMethod().**
Point 5: Constants	• Constants, which are values that do not change, are conventionally written in uppercase letters with underscores separating words. • For example: MAX_SIZE, PI.
Point 6: Module Names	• Module names should be short, lowercase, and descriptive. • Avoid using underscores in module names, as they are reserved for special meanings in Python.
Point 7: Private Names	• By convention, variables or functions intended for internal use within a module are prefixed with a single underscore. • This indicates that they are not intended to be accessed directly by users of the module.

Whitespace and Indentation

The **indentation** is how the Python interpreter knows which code belongs inside the **for** loop and which code belongs outside of it.

In Python, indentation is typically done using spaces or tabs (although spaces are recommended for consistency).

The standard convention is to use four spaces for each level of indentation.

Not all words are available for code.

These are words claimed or reserved by Python:

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

If you try **False = 'test'** or **for = 'test'** in a REPL, you will see that it is not possible to use them.

Comments

Comments are pieces of text that the computer does not look at when running a program. They are only meant for the human readers of the code.

There are two main types of comments in Python, Single-line and Multi-line comments:

Single-line comments start and end in the same line. It can stand alone, but can also be typed after the code (in this case we call it an **end-of-line comment**).
In both cases, we create them by placing a *# symbol* in front of our comment.

Multi-line comments are placed in multiple lines and provide more detailed explanations. We can write them by placing a *# symbol* at the beginning of each line, but we can also use a *triple quote " " "*.

Types and Casting

Data types specify the type of data that can be *stored inside a variable*.

If we assign a value to a variable, we can ask Python what the type of the value in that variable is. We can do this by using the built-in function **type()**, by placing the name of the variable in the brackets.

If you have a string with a number in it and you want to use it in a calculation, or if you have a number and want to use it in a sentence: that's possible.

You can **cast** the value to another type.

Casting means converting a value of one type to another type.

This is done by using *constructor functions*:

- **int()**
- **float()**
- **str()**

Strings

A string **(str)** is a fundamental data type that represents a sequence of characters.

You can declare a string using **single** ('), **double** (") and **triple double** (""") quotes.

For example:

```
example_one = ‘I am a string’
example_two = “Me too!”
example_three = “”” I too am a string
                        I am, in fact, a multiline string!””
```

String Operations

An **operator** is a character (or combination of characters) that signifies an operation that should be done on a number of **operands**.

For example, in Python (and every other major programming language) we can do *addition* with the **+** operator, which adds up the *operands* (the quantities on which an operation is to be done) to its left and right.

Note: If we try to perform the following operation: *'I like' + 3.14*

Python will give an error to let you know that adding a string and a number is not possible, but this issue can be solved with **casting.**

Membership Operator

We can use the membership operator **(in)** to test if a value is presented in a sequence, for example, in a string.

For example:

```
x = 'Samuel'
print(x in 'We went out for dinner with Anne, Samuel and Bob.')
```

What we can see is that after we assigned a *value (Samuel)* to a *variable (x),* the **in** operator in the second line checks if it appears in the object or not. In case it does, the output will be **True.**

Note: You cannot look for a number inside a string.

Comparing Values

To compare values **we use the == operator.**

It works for both **numbers** and **strings.** Depending on whether the values are the same or not, the operator will return *True* or *False*.

Indexing

By indexing, we refer to an element by its position, and we do it by writing the index number in between **square brackets [ ].**

⚠ **The first index is 0, not 1**

For example:

```
letter_grades = 'ABCDEF'
print(letter_grades[0])
print(letter_grades[3])
#now with negatives
print(letter_grades[-1])
print(letter_grades[-3])
```

Length

The built-in **len** function in Python is used to return the length (the number of items) of an object.

In case of a **string**, for example, len *returns the number of characters* in the string.

Interpolation

If we want to **insert a variable** into a string, we must then do the following:

- Write **f** *in front of the string*.
- Use curly braces (**{}**) to insert the variable into the string.
- Remember to **use quotes** to wrap the string (so add the first right after the f).

Boolean

The **boolean** is used to determine the truth value of an expression, and in Python, it can only have two values: **True** and **False** (mind the capitals!).

A boolean value can only be **True** or **False**, a helpful trick is to think of a name that says something which can be true or not.

For example: is_raining = True

Operators

To perform common mathematical operations, **we use arithmetic operators.**

EVENT	PARAMETER	EXAMPLE
+	Addition	3 + 2
-	Substaction	3 - 2
*	Multiplication	3 * 2
/	Division	3 / 2
%	Modulo	3 % 2
**	Exponentiation	2 ** 3
//	Floor division	3 / 2

Relational operators tell us something about the relation between two values.

OPERATOR	OPERATION
==	Equality
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

When we want to combine conditional statements, we use **logical operators.**

OPERATOR	OPERATION
and	True if both operands are true
or	True if either one of the operands are true
not	Negation. Reverses the truth value of its operand.