Setup

Python

PyCharm

Introduction

Keywords

- **Statement**
- **Comment**
- **Expression**
- **Function**
- **Parameter**
- **Variable**
- **Class**

Hello, world!

The very first program we will write is the more or less the traditional 'Hello world'. It will just print the favorite phrase of all software developers on the screen.

```
print("Hello world!")
print(1 + 1)
```

Now if after evaluation of the code we will see that

- we tell the computer what to do
- and `print` is a pre-defined code ready to be used by the developer
- and `print` uses a value to show it on the screen
- and the value used by print can be either pre-defined such as `'Hello, world!'`
- or take the result of operations such as `1 + 1`

If we elaborate further we will see that a program generally is a

- Sequence of commands called **statements**
- Each statement might consist of operations returning results called **expressions**
- Or each statement might use pre-defined code blocks called **functions**

- And each function may be configurable by the user through **parameters**

The final goal of a software product is then ...

- To work the intended way
- To be testable
- To be easily usable
- To be easily maintainable

Comments

The code can be documented using comments. Python offers single-line comments for quick clarifications beginning with the symbol `#`.

```
# This code will print a message and the sum of two numbers
print("Hello world!")
print(1 + 1)
```

There is also an option for multi-line comments for more elaborate explanations. The comment in this case must be enclosed in tipple quotes.

```
"""
This is a multi-line comment.

The following code will print the message `Hello, world!`
and then print the sum of two numbers.
"""

print("Hello world!")
print(1 + 1)
```

Declaring variables

Now let's modify our program a little bit to be more reusable. Now we make print to work with any text possible. For this purpose we will use a variable. A variable is a container for data, which can be reused many times in the program.

```
# The variable text holds `Hello, world!`
text = 'Hello world!'
print(text)
```

In the code above the variable `text` is declared and initialized. After this it may be used many times in the program. Now you might ask that this is absolutely unnecessary to use 2 lines of code instead of one. Let's give another example to demonstrate why variables are useful. Let's print 5 identical

messages on the screen ...

```
print("Hello world")
print("Hello world")
print("Hello world")
print("Hello world")
print("Hello world")
```

Now if we want to change the message on the screen we have to change all the statements in the program above. If we use a variable to store the text, the changes will be only in first statement.

```
text = 'Hello world!'
print(text)
print(text)
print(text)
print(text)
print(text)
```

Variable types

Python has several pre-defined variables types such as

- Strings which holds characters
- Numbers which hold integers, complex and floating point numbers
- Booleans which can be either true or false
- None used to signify that a variable is declared but not initialized

The following code will print all variables values and their types supported by Python at the time of speaking. Remember that the language is constantly evolving and it might be necessary to check the official documentation for changes. The type of the variable in Python can be checked by using `type()`.

```
# String
var = "Hello world!"
print(type(var), var)

# Integer number
var = 1
print(type(var), var)

# Real number
var = 3.14
print(type(var), var)

# Complex number
var = 1 + 1j
```

```
print(type(var), var)

# Boolean
var = False
print(type(var), var)

# Not initialized
var = None
print(type(var), var)
```

In the code above we can see that `print()` is configured with two parameters. The first one tells print to show the variable type on the screen and the second one the variable value. In Python `print()` supports an infinite amount of parameters. The output of the script is:

```
<class 'str'> Hello world!
<class 'int'> 1
<class 'float'> 3.14
<class 'complex'> (1+1j)
<class 'bool'> False
<class 'NoneType'> None
Process finished with exit code 0
```

We see the types of the variables, but they are preceeded by **class**. We defined earlier that variables are data containers, which can hold values. The **class** tells Python what type of data the variable stores and how Python can operate with the data.

The class concept can be demonstrated by first working with numbers of different types. In the example below Python analyses the code and deducts that a and b are integers and uses the integer class to check how to add them.

```
# The operator + adds two integer numbers
a = 1
b = 1
print(a + b)
```

In the same way in the next example the code is analysed Python and deducts that now a and b are complex numbers and thus another type of data. In this case Python will use the addition operation for this data class.

```
# The operator + adds two complex numbers
a = 1 + 1j
b = 1 + 1j
print(a + b)
```

Practice

1. Print your name on the screen
2. Make the program in such a way that you can print different names 10 times
3. Add an integer and a floating point number
4. Show the type of the result of the addition of floating point number and an integer
5. Check the type of `print`
6. Check the type of `type`
7. Add two integer, float and complex numbers and print the results

Operators

Arithmetic

The arithmetic operators are the most basic expressions in Python.

```
Addition        :   5 + 3   = 8         :   Addition of two numbers
Substraction    :   5 - 3   = 2         :   Substraction of two numbers
Multiplication  :   5 * 3   = 15        :   Multiplication of two numbers
Division        :   5 / 3   = 1         :   Float division of two numbers
Floor division  :   5 // 2  = 2         :   Integer division of two numbers
Modulus         :   5 % 3   = 2         :   Remainder after divison of two numbers
Exponentiation  :   5 ** 3  = 125       :   Exponentiation with base and exponent
```

Examples:

```
x = 5.0
y = 3.0
print(x / y)
print(x // y)
print(x % y)

print('-' * 80)

x = -5.0
y = 3.0
print(x / y)
print(x // y)
print(x % y)
```

Comparison

The comparision operators are used in expressions comparing different values. The result will be

either `True` or `False`.

```
Equal                       :   x == 1
Not equal                   :   x != 1
Greater than                :   x > 1
Greater than or equal to    :   x >= 1
Less than                   :   x < 1
Less than or equal to       :   x <= 1
Between                     :   1 < x < 2
```

Examples:

```
x = 1
y = 2
print(x > y)
print(x < y)
print(1 < x < 3)
```

Logical

The logical operators are used to control the flow of the program. The result will be either `True` or `False`.

```
NOT     :   not x
AND     :   x and y
OR      :   x or y
```

Examples:

```
x = 1
y = 2

print(not x)
print(x and y)
print(x or y)
```

Identity

The identity operators are used to compare objects and check if two objects are identical.

```
is          :   x is y      :   True if x and y are the same object
is not      :   x is not y  :   True if x and y are different objects
```

Membership

The membership operators are used check if an object is part of a collection of objects.

```
in          :   x in y      :   True if x is present in y
not in      :   x not in y  :   True if x is not present in y
```

Bitwise

The bitwise operators are used for setting individual bits in a byte, word and other integer types.

```
AND             :   x & y   :   1010 & 0101 = 0000, 1001 & 1111 = 1001
OR              :   x | y   :   1010 & 0101 = 1111, 1001 & 1111 = 1111
XOR             :   x ^ y   :   1010 & 0101 = 0000, 1001 & 1111 = 0110
NOT             :   ~x      :   ~1010 = 0101
Left shift      :   x << 1  :   1010 << 1 = 0100
Right shift     :   x >> 1  :   1010 >> 1 = 0101
```

```
x = int('1010', 2)
print(bin(x >> 1))
```

Conditional

The only ternary conditional statement in Python allows quick verification of a given condition and performing an action depending on the result of the check.

```
a, b = 10, 20

# Copy value of a in min if a < b else copy b
min = a if a < b else b
print(min)
```

Assignement

The assignement operator is used to change the value of a given variable. Python offers a variety of assignement operators, often combining arithmetic or bitwise operations and assignement in one statement.

```
x = 1       x = 1
x += 1      x = x + 1
x -= 1      x = x - 1
x *= 1      x = x * 1
x /= 1      x = x / 1
```

```
x %= 1      x = x % 1
x //= 1     x = x // 1
x **= 1     x = x ** 1
x &= 1      x = x & 1
x |= 1      x = x | 1
x ^= 1      x = x ^ 1
x >>= 1     x = x >> 1
x <<= 1     x = x << 1
```