Example 4.2.1. Creating and using Variables in Python

We use the basic IDLE (or another Python Editor) and type the following:

```
>>> x = 3
>>> x
3
```

Listing 4.2: Using Variables in Python

Here we define a variable and sets the value equal to 3 and then print the result to the screen.

[End of Example]

You can write one command by time in the IDLE. If you quit IDLE the variables and data are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script.

Python scripts or programs are save as a text file with the extension **.py**

Example 4.2.2. Calculations in Python

We can use variables in a calculation like this:

```
x = 3
y = 3*x
print(y)
```

Listing 4.3: Using and Printing Variables in Python

We can implementing the formula $y = ax + b$ like this:

```
a = 2
b = 5
x = 3

y = a*x + b

print(y)
```

Listing 4.4: Calculations in Python

As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

A variable can have a short name (like x and y) or a more descriptive name (sum, amount, etc).

You don need to define the variables before you use them (like you need to to in, e.g., C/C++/C).

Figure 4.1 show these examples using the basic IDLE editor.

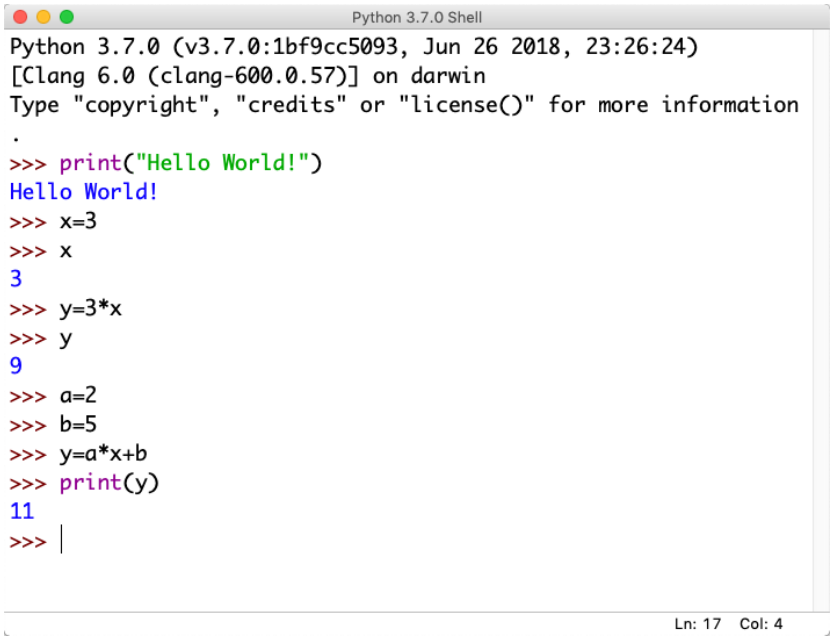

Figure 4.1: Basic Python

Here are some basic rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores
- Variable names are case-sensitive, e.g., amount, Amount and AMOUNT are three different variables.

4.2.1 Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them, so in normal coding you don't need to bother.

Example 4.2.3. Numeric Types in Python

```
x = 1       # int
y = 2.8     # float
z = 3 + 2j  # complex
```

Listing 4.5: Numeric Types in Python

This means you just assign values to a variable without worrying about what kind of data type it is.

```
print(type(x))
print(type(y))
print(type(z))
```

Listing 4.6: Check Data Types in Python

If you use the Spyder Editor, you can see the data types that a variable has using the Variable Explorer (Figure 4.2):

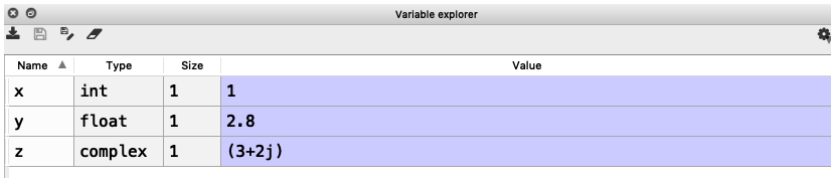

Figure 4.2: Variable Editor in Spyder

[End of Example]

4.2.2 Strings

Strings in Python are surrounded by either single quotation marks, or double quotation marks. 'Hello' is the same as "Hello".
Strings can be output to screen using the print function. For example: print("Hello").

Example 4.2.4. Plotting in Python

Below we see examples of using strings in Python:

```
a = "Hello World!"

print(a)

print(a[1])
print(a[2:5])
print(len(a))
print(a.lower())
```

```
print(a.upper())
print(a.replace("H", "J"))
print(a.split(" "))
```

Listing 4.7: Strings in Python

As you see in the example, there are many built-in functions form manipulating strings in Python. The Example shows only a few of them.

Strings in Python are arrays of bytes, and we can use index to get a specific character within the string as shown in the example code.

[End of Example]

4.2.3 String Input

Python allows for command line input.

That means we are able to ask the user for input.

Example 4.2.5. Plotting in Python

The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

Listing 4.8: String Input

[End of Example]

4.3 Built-in Functions

Python consists of lots of built-in functions. Some examples are the print(9 function that we already have used (perhaps without noticing it is actually a Built-in function).

Python also consists of different Modules, Libraries or Packages. These Modules, Libraries or Packages consists of lots of predefined functions for different topics or areas, such as mathematics, plotting, handling database systems, etc. See Section 4.4 for more information and details regarding this.

In another chapter we will learn to create our own functions from scratch.

4.4 Python Standard Library

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs.
The **Python Standard Library** consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.

The math module has all the basic math functions you need, such as: Trigonometric functions: *sin(x)*, *cos(x)*, etc. Logarithmic functions: *log()*, *log10()*, etc. Constants like *pi*, *e*, *inf*, *nan*, etc. etc.

Example 4.4.1. Using the math module

We create some basic examples how to use a Library, a Package or a Module:

If we need only the sin() function we can do like this:

```
from math import sin

x = 3.14
y = sin(x)

print(y)
```

If we need a few functions we can do like this

```
from math import sin, cos

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

If we need many functions we can do like this:

```
from math import *

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

We can also use this alternative:

```
import math

x = 3.14
y = math.sin(x)

print(y)
```

We can also write it like this:

```
import math as mt

x = 3.14
y = mt.sin(x)

print(y)
```

[End of Example]

There are advantages and disadvantages with the different approaches. In your program you may need to use functions from many different modules or packages. If you import the whole module instead of just the function(s) you need you use more of the computer memory.

Very often we also need to import and use multiple libraries where the different libraries have some functions with the same name but different use.

Other useful modules in the **Python Standard Library** are **statistics** (where you have functions like *mean()*, *stdev()*, etc.)

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

4.5 Using Python Libraries, Packages and Modules

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This approach has advantages and disadvantages. An disadvantage is that you need to install these packages separately and then later import these modules in your code.

Some important packages are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python
- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
- **Matplotlib** - Matplotlib is a Python 2D plotting library