



PYTHON BASICS



Ragesh Hajela

Senior Programmer, Amadeus Labs

Contact : +91-9620121987, rageshhajela@gmail.com

Training Syllabus

1. Introductory Sessions

- a. History of Python
- b. Introduction
- c. Starting with Python
- d. Execute python script

2. Basic Data Types

- a. Indentation
- b. Data Types and Variables
- c. Operators
- d. Lists and Strings
- e. List Manipulations
- f. Shallow and Deep Copy

3. Advanced Data Types

- a. Dictionaries
- b. Set and Frozen Sets
- c. Tuple
- d. Input via the keyboard

4. Conditional Statements

- a. If-else Block
- b. Loops, While Loop
- c. For Loop
- d. Iterators and Iterables

5. Output Formatting

- a. Output with Print**
- b. String Modulo**
- c. Format Method**

6. Functions

- a. Introduction to Functions**
- b. Recursion and Recursive Functions**
- c. Parameter Passing in Functions**
- d. Namespaces**
- e. Global and Local Variables**
- f. Decorators**

7. File Operations

- a. Read and Write Files**
- b. Modules**
- c. Packages**

8. Advanced Python Concepts

- a. Regular Expressions**
- b. Lambda, Filter & Map Operators**
- c. List Comprehension**
- d. Iterators and Generators**
- e. Exception Handling**
- f. Tests, DocTests & UnitTests**

9. Object Oriented Programming

- a. Introduction
- b. Class and Instance Attributes
- c. Properties v/s getters and setters
- d. Inheritance
- e. Classes and Class Creation
- f. Abstract Classes

10. Powerful Utilities

- a. Multithreading
- b. Logging Facility
- c. Argument Parser
- d. Invoking Sub process
- e. Lincache, fnmatch etc.
- f. Live Project

11. Domain Specific

- a. Networking Automations, or
- b. Django Web Framework, or
- c. Numerical Data Analytics
 - i. Numpy
 - ii. Pandas, or
- d. Data Science Analytics
 - i. Web Scrapping & Crawling
 - ii. Machine Learning, or
- e. Automation Tooling

Topic 1a: History of Python

Easy as ABC

- What do the alphabet and the programming language Python have in common? Right, both start with ABC.
- ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde & Informatica). The greatest achievement of ABC was to influence the design of Python.
- Python was conceptualized in the late 1980s.
- Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system.
- He experienced many frustrations with ABC and decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems.
- He created a simple virtual machine, a simple parser, and a simple runtime. He made his own version of the various ABC parts that he liked.
- He created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of

powerful data types: a hash table (or dictionary), a list, strings, and numbers.

What about the name “Python”?

- Most people think about snakes, and even the logo depicts two snakes, but the origin of the name has its root in British humour.
- During Christmas vacation in 1989, he decided to write an interpreter for the new scripting language: a descendant of ABC.
- He chose Python as a working title for a “hobby” project, being in a slightly irreverent mood during Christmas’1989 (and a big fan of Monty Python's Flying Circus).

Development Steps of Python?

- Rossum published the first version of Python code in February 1991.
- Python version 1.0 was released in January 1994.
- In October 2000, Python 2.0 was introduced.
- Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 was released in 2008.

- Python 3 is not backwards compatible with Python 2.x
- Some changes in Python 3.0:
 - Print is now a function
 - Views and iterators instead of lists
 - The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behavior.
 - raw_input() in Python 2.x is changed to input() & the former is discontinued.
 - input() in Python 2.x is changed to eval(input)

Topic 1b: Introduction

Features

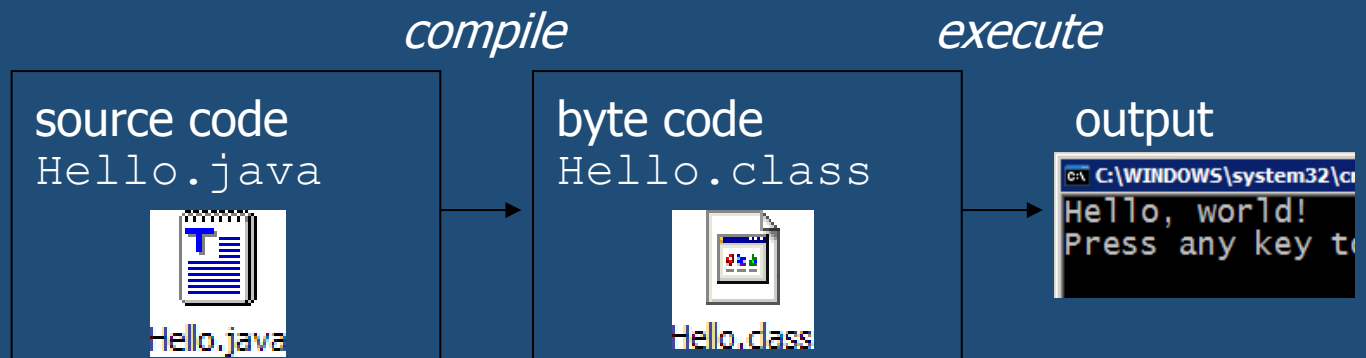
- Multi-purpose (Web, Data Science, Automation, etc.)
- Procedural and Object-Oriented
- Interpreted
- Interactive Shell
- Strongly typed and Dynamically typed
- Focus on readability and productivity
- Cross Platform (CPython, Jython, IronPython, PyPy)

Few Big Organizations using Python

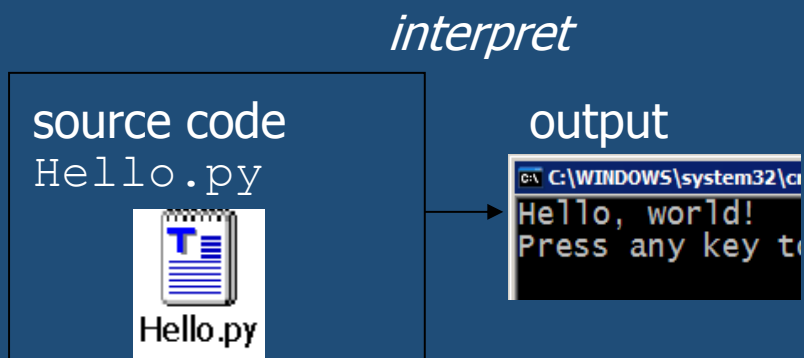
- Google
- NASA
- Netflix
- Dropbox
- YouTube
- New York University
- General Electric
- Juniper Networks
- Lego
- Nasdaq
- And the list goes on...

Compiled v/s Interpreted Language

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly *interpreted* into machine instructions.



That's why Python is an interpreted language and not a compiled one. But, more insights on this topic will come in Topic 1d.

Topic 1c: Starting with Python

The Interpreter, an Interactive Shell

- The interactive shell is between the user and the operating system (e.g. Linux, Unix, Windows or others)
- The Python interpreter can be used from an interactive shell.
- The interactive shell is also interactive in the way that it stands between the commands or actions and their execution
- Python offers a comfortable command line interface with the Python shell, which is also known as the "Python interactive shell".

Setup

- We can access Python in four ways:
 - Download & install Python latest version from <https://www.python.org/downloads/>, and then launch IDLE. It is the interactive development environment for Python. We can access it from Windows Search Menu.
 - We can also launch Python – Command line directly after installation, instead of IDLE

- The Python interpreter can also be invoked by launching windows command prompt & typing the command "python" without any parameter followed by the "return" key at the shell prompt:

```
python
```

Python comes back with the following information

```
$ python
Python 2.7.11+ (default, Apr 17
2016, 14:00:29)
[GCC 5.3.1 20160413] on linux2
Type "help", "copyright",
"credits" or "license" for more
information.
>>>
```

This will need to set the environment variable in Advanced System Settings.

- Finally, other way is to download a Python IDE (Interactive Development Environment) such as PyCharm, Jupiter etc. These also have facilities for debugging a program using breakpoints.
- Recommended:
 - We will use IDLE for initial classes and then move to PyCharm IDE.

Let's try some simple commands

```
>>> hello
Traceback (most recent call
last):
  File "<stdin>", line 1, in
<module>
NameError: name 'hello' is not
defined
>>>
```

```
>>> print("Hello World")
Hello World
>>>
```

```
>>> "Hello World"
'Hello World'
>>> 3
3
>>>
```

- How to quit the Python Shell?
 - `Exit()`, or
 - `Quit()`

Let's try a simple calculator

```
>>> 4.567 * 8.323 * 17
646.18939699999999
>>>
```

- Python follows the usual order of operations in mathematical expressions.
- The standard order of operations is expressed in the following enumeration:
 - exponents and roots
 - multiplication and division
 - addition and subtraction
- This means that we don't need parenthesis in the expression "3 + (2 * 4)"

```
>>> 3 + 2 * 4
11
>>>
```

- The most recent output value is automatically stored by the interpreter in a special variable with the name "_" and can be used in other expressions like any other variable

```
>>> _ * 3
33
>>>
```

- The underscore variable is only available in the Python shell. It's NOT available in Python scripts or programs.

Let's try Basic Variables & Strings

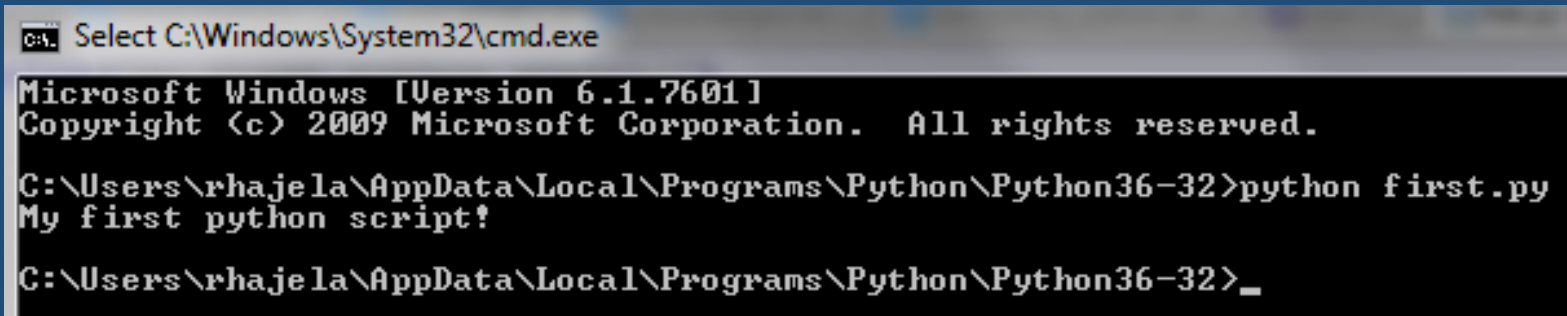
```
>>> maximal = 124
>>> width = 94
>>> print(maximal - width)
30
>>>
```

```
>>> "Hello" + " " + "World"
'Hello World'
```

```
>>> ".-." * 4
'-.-.-.-.-.-.-'
>>>
```

Topic 1d: Execute python script

First Python Script

A screenshot of a Windows command prompt window. The title bar reads "Select C:\Windows\System32\cmd.exe". The window content shows the Microsoft Windows version (6.1.7601) and copyright information. The current directory is C:\Users\rhajela\AppData\Local\Programs\Python\Python36-32. The user has entered the command "python first.py" and the output is "My first python script!". The prompt is now waiting for the next command.

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\rhajela\AppData\Local\Programs\Python\Python36-32>python first.py
My first python script!

C:\Users\rhajela\AppData\Local\Programs\Python\Python36-32>_
```

```
print("My first python script!")
```

Python Internals

- As we have studied earlier that Python language is an interpreted programming or a script language.
- The truth is: Python is both an interpreted and a compiled language. But calling Python a compiled language would be misleading.
- What is a compiler?
 - A compiler is a computer program that transforms (translates) source code of a programming language (the source language) into another computer language (the target language), mostly assembly or machine code.
 - An interpreter is a computer program that executes instructions written in a programming language. It can either

- execute the source code directly or
- translates the source code in a first step into a more efficient representation and executes this code.
- People would assume that the compiler translates the Python code into machine language. Python code is translated into intermediate code, which has to be executed by a virtual machine, known as the PVM, the Python virtual machine.
- There is even a way of translating Python programs into Java byte code for the Java Virtual Machine (JVM). This can be achieved with Jython.
- Question: Do we have to compile our Python scripts to make them faster or how can we compile them? The answer is easy: No, you don't need to do anything because "Python" is doing the thinking for you automatically.
- But, if still we want to compile, can be achieved.

```
>>> import py_compile
>>>
py_compile.compile('my_first_simple_script.py')
>>>
```

Or

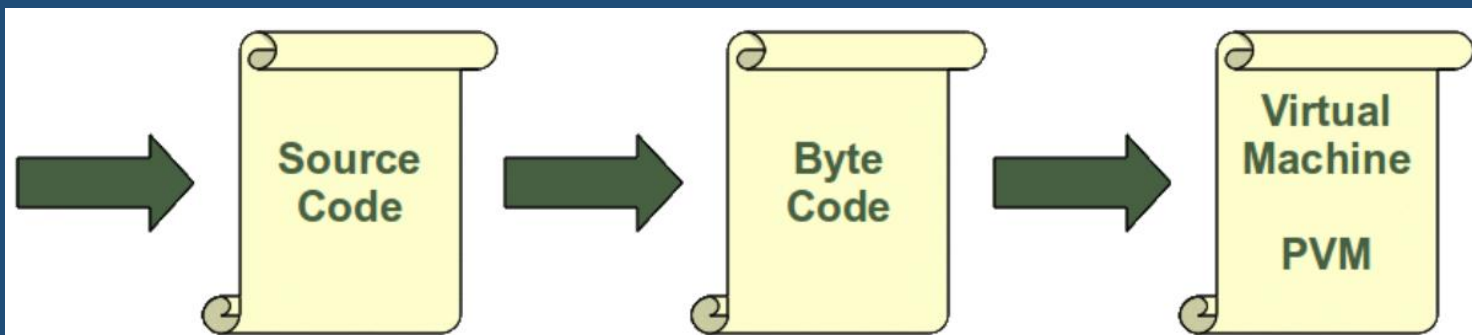

```
python -m py_compile  
my_first_simple_script.py
```

- As a result, there will be a new subdirectory "__pycache__" created, if it hasn't already existed. And also find a file "my_first_simple_script.cpython-34.pyc" in this subdirectory. This is the compiled version of our file in byte code.

- We can also compile all scripts as this:

```
monty@python:~/python$ python -  
m compileall .  
Listing . ...
```

- How does .pyc file generate? - If Python has write-access for the directory where the Python program resides, it will store the compiled byte code in a file that ends with a .pyc suffix. If Python has no write access, the program will work anyway. The byte code will be produced but discarded when the program exits.



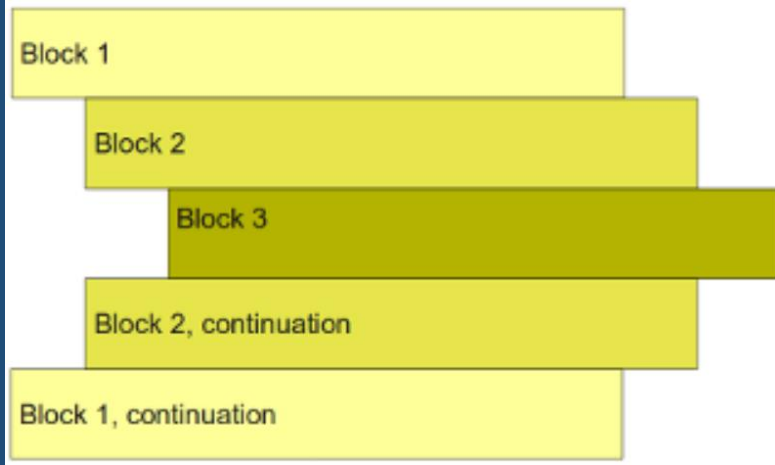
Topic 2a: Indentation

Blocks

- A block is a group of statements in a program or script. Usually it consists of at least one statement.
- A language, which allows grouping with blocks, is called a block structured language.
- Generally, blocks can contain blocks as well, so we get a nested block structure.
- A block in a script or program functions as a mean to group statements to be treated as if they were one statement.
- The first time, block structures had been formalized was in ALGOL, called a compound statement.
- Programming languages usually use certain methods to group statements into blocks:
 - Begin ... end : ALGOL, Pascal etc.
 - do ... done : Bourne and Bash Shell etc
 - Braces { ... } : C, C++, Java, Perl

Indenting Code

- Python uses a different principle. Python programs get structured through indentation, i.e. code blocks are defined by their indentation.



- In Python, it's a mandatory language requirement, not a matter of style.
- This principle makes it easier to read and understand other people's Python code.
- So, how does it work? All statements with the same distance to the right belong to the same block of code.
- If a block has to be more deeply nested, it is simply indented further to the right.
- Example:

```
print("Start")
a = 1
if a == 1:
    print("Step1")
    b = 3
    if b == 2:
        print("step2")
    else:
        print("step3")
    print("step4")
print("end")
```

Topic 2b: Data Types & Variables

Basic Difference from other languages

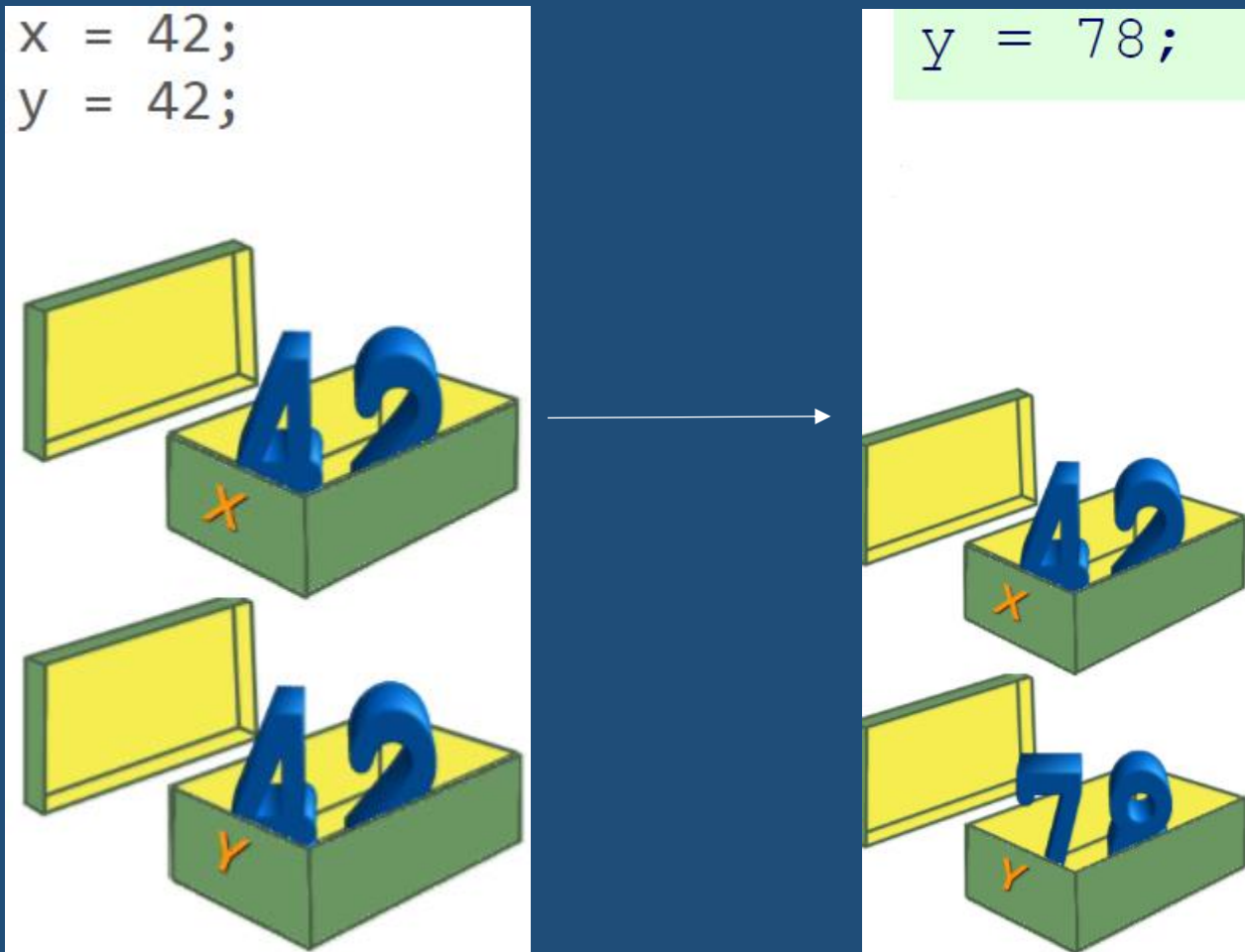
If you want to use lists or an arrays in C or C++, you will have to construe the data type list or associative arrays from scratch, i.e. design memory structure and the allocation management. You will have to implement the necessary search and access methods as well.

But, Python provides power data types like lists and associative arrays, called dict in Python, as a genuine part of the language.

General Concept of Variables in other languages

- What is a variable? As the name implies, a variable is something, which can change.
- A variable is a way of referring to a memory location used by a computer program.
- In many programming languages, a variable is a symbolic name for this physical location.
- This memory location contains values, like numbers, text or more complicated types.

- We can use this variable to tell the computer to save some data in this location or to retrieve some data from this location.
- So, a variable can be seen as a container to store data.
- These concepts about variables fits best to the way variables are implemented in C, C++ or Java.



- Such declarations make sure that the program reserves memory for two variables (memory locations) with the names x and y. Putting values into the variables can be realized with assignments.
- Variables should have unique data type. E.g. if a variable is of type integer, solely integers can be saved in the variable during the duration of the program.

Variables in Python

- It's a lot easier in Python. There is no declaration of variables required in Python.
- If there is need of a variable, you think of a name and start using it as a variable.
- Another remarkable aspect of Python: Not only the value of a variable may change during program execution but the type as well.
- You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the same variable.
- In this line, we assign the value 42 to a variable:

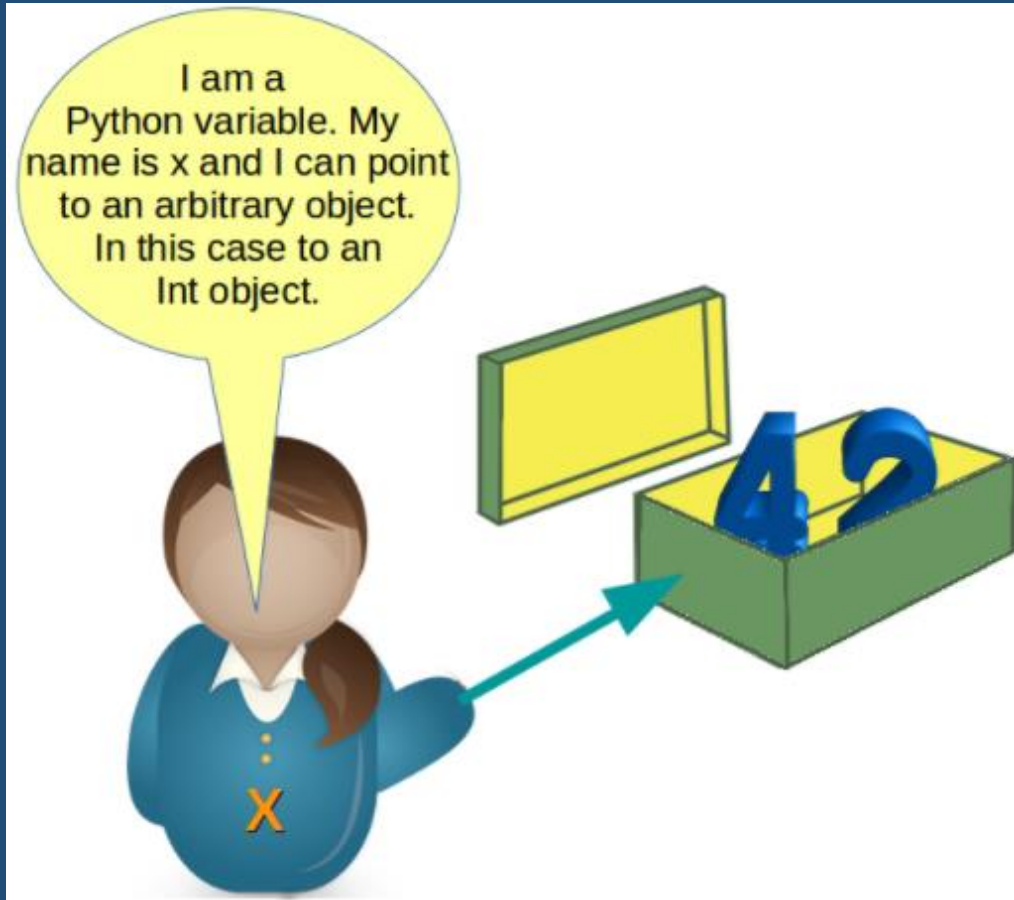
$$i = 42$$

- The equal "=" sign in the assignment shouldn't be seen as "is equal to". It should be read as "is set to", meaning in our example "the variable i is set to 42".

```
i = 42 # data
type is implicitly set to
integer
i = 42 + 0.11 # data
type is changed to float
i = "forty" # and
now it will be a string
```

Object References

Python variables are references to objects, but the actual data is contained in the objects.

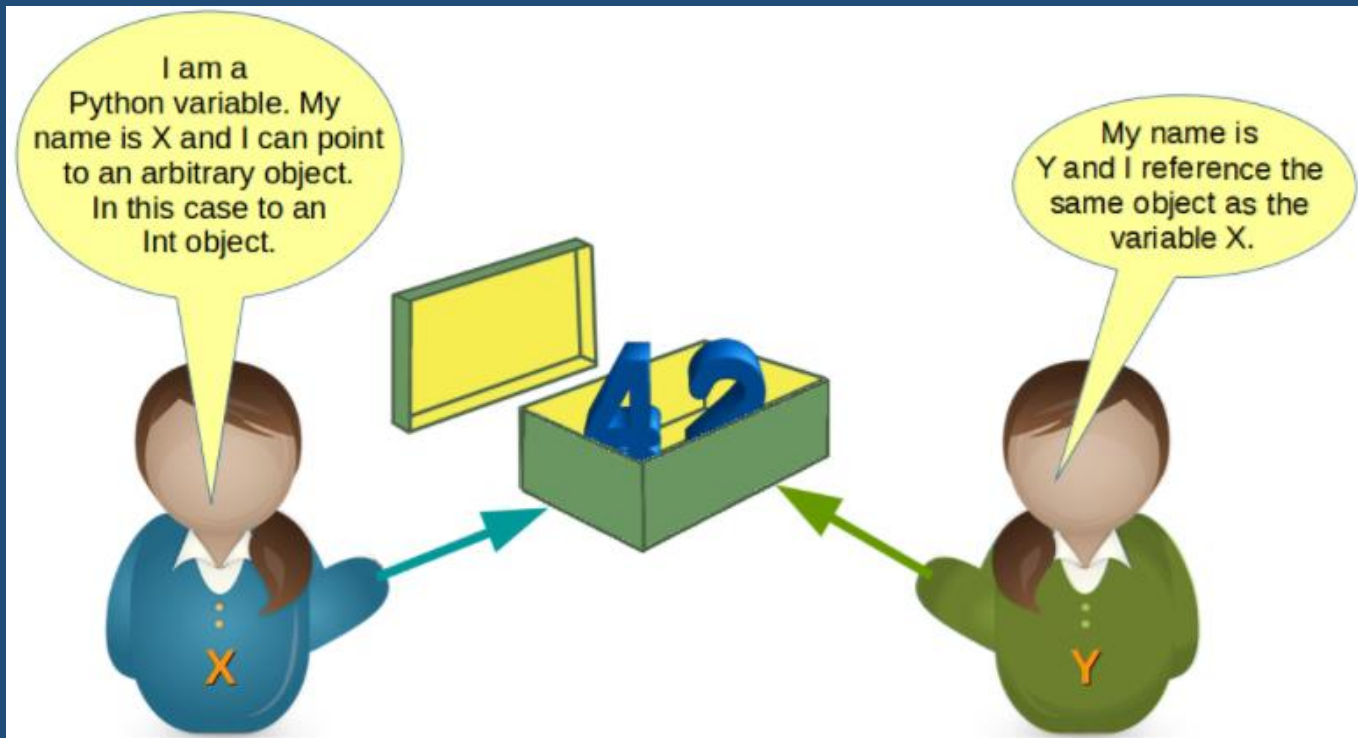


As variables are pointing to objects and objects can be of arbitrary data type, variables can't have types associated with them. This is a huge difference to C, C++ or Java, where a variable is associated with a fixed data type.

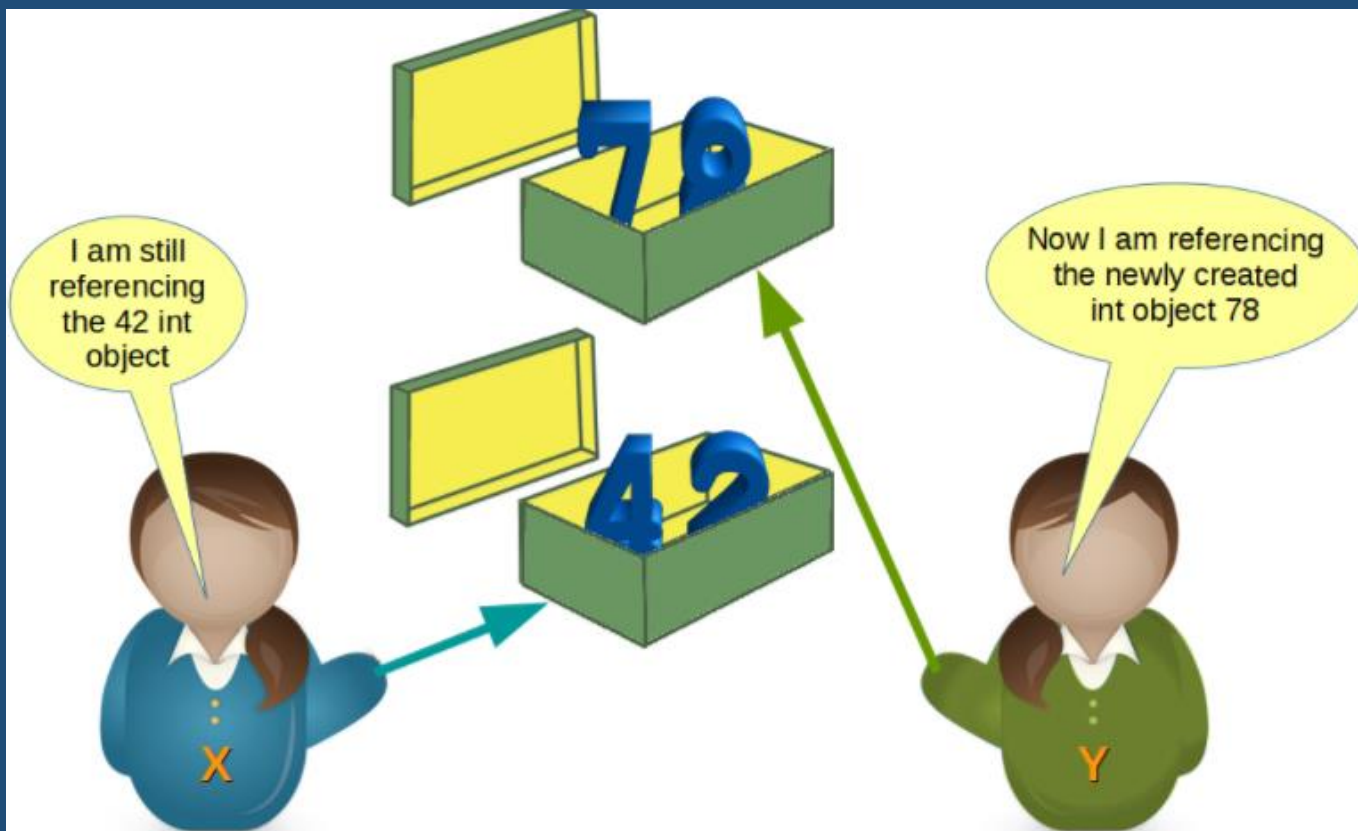
```
>>> x = 42
>>> print(x)
42
>>> x = "Now x references a string"
>>> print(x)
Now x references a string
```


Examples

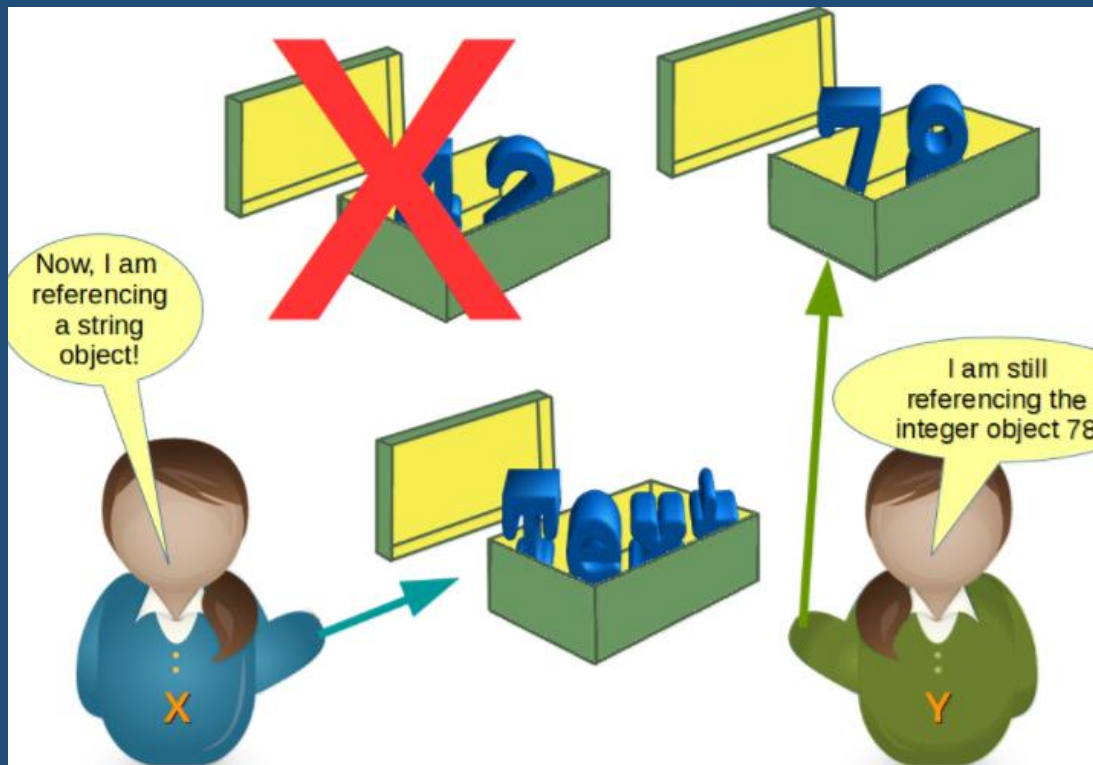
```
>>> x = 42  
>>> y = x
```



```
y = 78
```



Now, let us do a string assignment to variable x.



id function as a proof

The identity function `id()` can be used for this purpose. Every instance (object or variable) has an identity, i.e. an integer that is unique within the script or program, i.e. other objects have different identities.

```
>>> x = 42
>>> id(x)
10107136
>>> y = x
>>> id(x), id(y)
(10107136, 10107136)
>>> y = 78
>>> id(x), id(y)
(10107136, 10108288)
>>>
```

Valid Variable Names

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- A variable name and an identifier can consist of the uppercase letters "A" through "Z", the lowercase letters "a" through "z", the underscore _ and, except for the first character, the digits 0 through 9.
- Identifiers are unlimited in length. Case is significant.

Python Keywords

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

```
>>> help()
```

```
Welcome to Python 3.6's help utility!
```

```
If this is your first time using Python, you should definitely check out  
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing  
Python programs and using Python modules. To quit this help utility and  
return to the interpreter, just type "quit".
```

```
To get a list of available modules, keywords, symbols, or topics, type  
"modules", "keywords", "symbols", or "topics". Each module also comes  
with a one-line summary of what it does; to list the modules whose name  
or summary contain a given string such as "spam", type "modules spam".
```

Numbers

There are four built-in data types for numbers:

- *Integer*

- Normal Integers e.g. 4321
- Octal Literals (base 8): A number prefixed by 0o (zero and a lowercase "o" or uppercase "O") will be interpreted as an octal number

```
>>> a = 0o10
>>> print(a)
8
```

- Hexadecimal literals (base 16): Hexadecimal literals have to be prefixed either by "0x" or "0X"

```
>>> hex_number = 0xA0F
>>> print(hex_number)
2575
```

- Binary literals (base 2): Binary literals can easily be written as well. They have to be prefixed by a leading "0", followed by a "b" or "B"

```
>>> x = 0b101010 >>> x 42 >>>
```

- The functions hex, bin, oct can be used to convert an integer number into the corresponding string representation of the integer number

```

>>> x = hex(19)
>>> x
'0x13'
>>> type(x)
<class 'str'>
>>> x = bin(65)
>>> x
'0b1000001'
>>> x = oct(65)
>>> x
'0o101'
>>> oct(0b101101)
'0o55'
>>>

```

- **Integers in Python3 can be of unlimited size**

```

>>> x =
78736609871273890324567823478235829283749
8729182728
>>> print(x)
78736609871273890324567823478235829283749
8729182728
>>> x * x * x
48812397007063821598677016210573131553882
75860919486179978711229502288911239609019
18308618286311523282239313708275589787123
005317148968569797875581092352
>>>

```

- ***Long Integers:*** In Python3 there is only one "int" type, which contains both "int" and "long" from Python2.

- **Floating-point numbers.** For example, 42.11
- **Complex Numbers:** Complex numbers are written as <real part> + <imaginary part>j

```
>>> x = 3 + 4j
>>> y = 2 - 3j
>>> z = x + y
>>> print(z)
(5+1j)
```

Integer Division

There are two kinds of division operators:

1. "true division" performed by "/"

```
>>> 10 / 3
3.3333333333333335
>>> 10.0 / 3.0
3.3333333333333335
>>> 10.5 / 3.5
3.0
```

2. "floor division" performed by "//"

```
>>> 9 // 3
3
>>> 10 // 3
3
>>> 11 // 3
3
>>> 12 // 3
4
>>> 10.0 // 3
3.0
>>> -7 // 3
-3
>>> -7.0 // 3
-3.0
```

Strings

There are different ways to define strings in Python:

```
>>> s = 'I am a string enclosed in single quotes.'  
>>> s2 = "I am another string, but I am enclosed in double quotes"
```

Both `s` and `s2` of the previous example are variables referencing string objects.

Examples:

```
>>> s3 = 'It doesn\'t matter!'
```

```
>>> s3 = "It doesn't matter!"
```

```
>>> txt = "He said: \"It doesn't matter, if you enclose a string in single or double quotes!\""
```

```
txt = '''A string in triple quotes can extend over multiple lines like this one, and can contain 'single' and "double" quotes.'''
```

A string in Python consists of a series or sequence of characters - letters, numbers, and special characters. Strings can be subscripted or indexed.

-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
>>> s = "Hello World"
>>> s[0]
'H'
>>> s[5]
' '
```

```
>>> s[-1]
'd'
>>> s[-2]
'l'
```

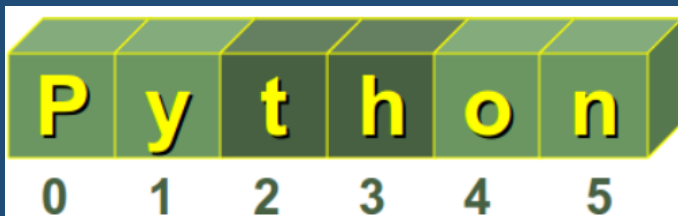
```
>>> s[len(s)-1]
'd'
```

Python strings cannot be changed. Trying to change an indexed position will raise an error:

```
>>> s = "Some things are immutable!"
>>> s[-1] = "."
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Some Operators and functions for strings

- **Concatenation:**
"Hello" + "World" will result in "HelloWorld"
- **Repetition:**
"*_*" * 3 will result in "*_**_**_**_**"
- **Indexing:**
"Python"[0] will result in "P"
- **Slicing:** "Python"[2:4] will result in "th"



- **Size:**
len("Python") will result in 6

A String Peculiarity

- If both `a` and `b` are strings, "`a is b`" checks if they have the same identity, i.e. share the same memory location.
- If "`a is b`" is `True`, then it trivially follows that "`a == b`" has to be `True` as well.
- Yet, "`a == b`" `True` doesn't imply that "`a is b`" is `True` as well!

Examples:

```
>>> a = "Linux"
>>> b = "Linux"
>>> a is b
True
```

```
>>> a = "Baden!"
>>> b = "Baden!"
>>> a is b
False
```

```
>>> a == b
True
```

```
>>> a = "Baden1"
>>> b = "Baden1"
>>> a is b
True
```

Escape Sequences in Strings

The backslash (`\`) character is used to escape characters, i.e. to "escape" the special meaning, which this character would otherwise have.

<code>\\</code>	Backslash (<code>\</code>)		
<code>\'</code>	Single quote (<code>'</code>)		
<code>\"</code>	Double quote (<code>"</code>)	<code>\n</code>	ASCII Linefeed (LF)

Topic 2c: Operators

Arithmetic and Comparison Operators

These are the various built-in operators, which Python has to offer for any operation on numeric types.

Operator	Description	Example
<code>+, -</code>	Addition, Subtraction	<code>10 - 3</code>
<code>*, %</code>	Multiplication, Modulo	<code>27 % 7 -> Result: 6</code>
<code>/</code>	Division	<code>10 / 3 -> Result: 3</code>
<code>//</code>	Floor Division The results of <code>int(10 / 3)</code> and <code>10 // 3</code> are equal. But the <code>"//"</code> division is more than two times as fast! <div><pre>import timeit print(timeit.timeit(('for x in range(1, 100):y =100 // x', number=100000))</pre></div>	<code>10 // 3 -> Result: 3</code> <code>10.0 // 3 ->Result: 3.0</code>
<code>+x, -x</code>	Unary Minus and Unary Plus (Algebraic signs)	<code>-3</code>
<code>**</code>	Exponentiation	<code>10 ** 3 ->Result: 1000</code>
<code>or, and, not</code>	Boolean Or, And, Not	<code>(a or b) and c</code>
<code>in</code>	"Element of"	<code>1 in [3, 2, 1]</code>
<code><, <=, >, !=</code>	Comparison operator	<code>2 <= 3</code>

Topic 2d: Lists and Strings

Introduction to Lists

A list in Python is an ordered group of items or elements. These list elements don't have to be of the same type. It can be an arbitrary mixture of elements like numbers, strings, other lists and so on.

The main properties of Python lists:

- They are ordered
- They contain arbitrary objects
- Elements of a list can be accessed by an index
- They are arbitrarily nestable, i.e. they can contain other lists as sublists
- Variable size
- They are mutable, i.e. the elements of a list can be changed

List	Description
<code>[]</code>	An empty list
<code>[1,1,2,3,5,8]</code>	A list of integers
<code>[42, "What's the question?", 3.1415]</code>	A list of mixed data types
<code>["Stuttgart", "Freiburg"]</code>	A list of strings

<code>[["London","England", 7556900], ["Paris","France",219 3031]]</code>	A nested list
<code>["High up", ["further down", ["and down", ["deep down", "the answer", 42]]]]</code>	A deeply nested list

Accessing List elements

```
>>> languages = ["Python", "C", "C++", "Java",  
"Perl"]  
>>> print(languages[0] + " and " +  
languages[1] + " are quite different!")  
Python and C are quite different!  
>>> print("Accessing the last element of the  
list: " + languages[-1])  
Accessing the last element of the list: Perl
```

Sublists

```
>>> person = [["Marc","Mayer"],["17, Oxford  
Str", "12345","London"],"07876-7876"]  
>>> name = person[0]  
>>> print(name)  
['Marc', 'Mayer']  
>>> first_name = person[0][0]  
>>> print(first_name)  
Marc  
>>> last_name = person[0][1]  
>>> print(last_name)  
Mayer  
>>> address = person[1]  
>>> street = person[1][0]  
>>> print(street)  
17, Oxford Str
```

```
>>> complex_list = [["a", ["b", ["c", "x"]]]]
>>> complex_list = [["a", ["b", ["c", "x"]]], 42]
>>> complex_list[0][1]
['b', ['c', 'x']]
>>> complex_list[0][1][1][0]
'c'
```

Tuples

A tuple is an immutable list, i.e. a tuple cannot be changed in any way once it has been created.

A tuple is similar to lists, except that the set of elements is enclosed in parentheses instead of square brackets.

Where is the benefit of tuples?

- Tuples are faster than lists.
- Knowing that some data doesn't have to be changed, we should use tuples instead of lists, because this protects our data against accidental changes.
- Main advantage is that tuples can be used as keys in dictionaries, while lists can't.

```
>>> t = ("tuples", "are", "immutable")
>>> t[0]
'tuples'
>>> t[0]="assignments to elements are not possible"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Slicing

To extract part of a string or a list, we can use the slice operator ":" in Python

```
>>> str = "Python is great"
>>> first_six = str[0:6]
>>> first_six
'Python'
>>> starting_at_five = str[5:]
>>> starting_at_five
'n is great'
>>> a_copy = str[:]
>>> without_last_five = str[0:-5]
>>> without_last_five
'Python is '
```

```
>>> cities = ["Vienna", "London", "Paris",
"Berlin", "Zurich", "Hamburg"]
>>> first_three = cities[0:3]
>>> # or easier:
...
>>> first_three = cities[:3]
>>> print(first_three)
['Vienna', 'London', 'Paris']
```

```
>>> all_but_last_two = cities[:-2]
>>> print(all_but_last_two)
['Vienna', 'London', 'Paris', 'Berlin']
```

Slicing works with three arguments as well. If the third argument is for example 3, only every third element of the list, string or tuple from the range of the first two arguments will be taken.

```
s[begin: end: step]
```

Example

```
>>> str = "Python under Linux is great"
>>> str[::3]
'Ph d n e'
```

```
>>> s = "Toronto is the largest City in Canada"
>>> t = "Python courses in Toronto by Bodenseo"
>>> s = "".join(["".join(x) for x in zip(s,t)])
>>> s
'TPoyrtohnnotno ciosu rtshees lianr gTeosrto
nCtiot yb yi nB oCdaennasdeao'
```

```
>>> s
'TPoyrtohnnotno ciosu rtshees lianr gTeosrto
nCtiot yb yi nB oCdaennasdeao'
>>> s[::2]
'Toronto is the largest City in Canada'
>>> s[1::2]
'Python courses in Toronto by Bodenseo'
```

Length

The length of a sequence, i.e. a list, a string or a tuple, can be determined with the function `len()`.

For strings it counts the number of characters and for lists or tuples the number of elements are counted, whereas a sublist counts as 1 element.

```
>>> txt = "Hello World"
>>> len(txt)
```

```
11
>>> a = ["Sven", 45, 3.54, "Basel"]
>>> len(a)
4
```

Concatenation of Sequences

```
>>> firstname = "Homer"
>>> surname = "Simpson"
>>> name = firstname + " " + surname
>>> print(name)
Homer Simpson
```

```
>>> colours1 = ["red", "green", "blue"]
>>> colours2 = ["black", "white"]
>>> colours = colours1 + colours2
>>> print(colours)
['red', 'green', 'blue', 'black', 'white']
```

augmented assignment `"+="` e.g. `s += t` → `s = s + t`

Checking if an element is in list

```
>>> abc = ["a", "b", "c", "d", "e"]
>>> "a" in abc
True
>>> "a" not in abc
False
>>> "e" not in abc
False
>>> "f" not in abc
True
>>> str = "Python is easy!"
>>> "y" in str
True
>>> "x" in str
False
```

Repetitions

"*" operator is a kind of abbreviation for an n-times concatenation i.e. $\text{str} * 4 \rightarrow \text{str} + \text{str} + \text{str} + \text{str}$

```
>>> 3 * "xyz-"
'xyz-xyz-xyz-'
>>> "xyz-" * 3
'xyz-xyz-xyz-'
>>> 3 * ["a", "b", "c"]
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

The augmented assignment for **"*"** can be used as well:
 $s *= n$ is the same as $s = s * n$.

The Pitfalls of Repetitions

In our last example, we applied the repetition operator on strings and flat lists. Let us apply it to nested lists as well:

```
>>> x = ["a", "b", "c"]
>>> y = [x] * 4
>>> y
[['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c'], ['a', 'b', 'c']]
>>> y[0][0] = "p"
>>> y
[['p', 'b', 'c'], ['p', 'b', 'c'], ['p', 'b', 'c'], ['p', 'b', 'c']]
```

We have assigned a new value to the first element of the first sublist of `y`, i.e. `y[0][0]` and we have "automatically" changed the first elements of all the sublists in `y`, i.e. `y[1][0]`, `y[2][0]`, `y[3][0]`.

The reason is that the repetition operator `"* 4"` creates 4 references to the list `x`: thus every element of `y` is changed, if we apply a new value to `y[0][0]`.

