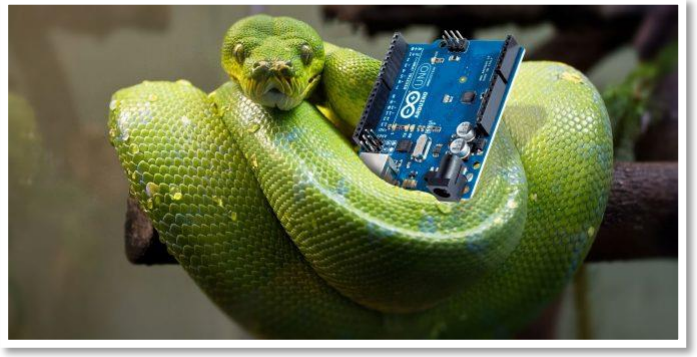
What is Python?

- An object-oriented, high-level programming language
- Used for wide variety of applications and not limited to basic usage
- A great language for beginners because of its readability and other structural elements designed to make it easy to understand



It was created in 1991 by Guido van Rossum

The Python programming language was considered a gap-filler, a way to write scripts that “automate the boring stuff”

Over the past few years, Python has emerged as a first-class citizen in modern software development, infrastructure management, and data analysis

It is no longer a back-room utility language, but a major force in web application creation and systems management, and a key driver of the explosion in big data analytics and machine intelligence.

Advantages & Disadvantages

Benefits

- Extensive Libraries
- Extensible
- Embeddable
- Improved Productivity
- IOT Opportunities
- Simple & Easy
- Readable
- Object-Oriented
- Free and Open source
- Portable

Limitations

- Speed Limitations
- Weak in mobile computing and browsers
- Design Restrictions
- Underdeveloped database access layers
- Simple
- Run time errors

Benefits

Extensive Libraries	Python downloads with extensive library
	These contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, and more
Extensible	Python can be extended to other languages
	Write some of the code in languages like C++ or C
Embeddable	Put the Python code in source code of a different language, like C++
	Adds scripting capabilities to code in the other language
Improved Productivity	The language's simplicity and extensive libraries render programmers more productive
	The fact that you need to write less lets more get done
IoT Opportunities	Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for Internet Of Things.
	This is a way to connect the language with the real world
Simple & Easy	Easy to learn, understand and code
Readable	It is not a verbose language hence easy to read like reading English
Object Oriented	This language supports both the procedural and object-oriented programming paradigms
Free & Open Source	Python is freely available.
	Also download its source code, make changes to it, and even distribute it.
Portable	Need to code only once, and can run it anywhere.
	This is called Write Once Run Anywhere (WORA)

Limitations

Speed Limitations	Python is executed line by line, it often results in slow execution
Weak in Mobile Computing & Browsers	Python is much rarely seen on the client-side. It is rarely ever used to implement smartphone-based applications as it isn't that secure
Design Restrictions	Python is dynamically typed and can raise run time errors
Underdeveloped Database Access Layers	Python's database access layers are a bit underdeveloped compared to more widely used technologies like JDBC, ODBC
Simple	Python's simplicity can indeed be a problem

Running Python Scripts

Executing Scripts with Python Launcher

The Python launcher for Windows is a utility which aids in the location and execution of different Python versions. It allows scripts (or the command-line) to indicate a preference for a specific Python version, and will locate and execute that version

- **From the command-line**

You should ensure the launcher is on your PATH - depending on how it was installed it may already be there, but check just in case it is not.

From a command-prompt, execute the following command:

```
py
```

- **From a script**

Let's create a test Python script - create a file called `hello.py` with the following contents

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

```
py hello.py
```

- **From file associate**

The launcher should have been associated with Python files (i.e. .py, .pyw, .pyc, .pyo files) when it was installed.

Executing Scripts without Python Launcher

- Python scripts (files with the extension .py) will be executed by python.exe by default
- This executable opens a terminal, which stays open even if the program uses a GUI.
- If you do not want this to happen, use the extension .pyw which will cause the script to be executed by pythonw.exe by default (both executables are located in the top-level of your Python installation directory).
- This suppresses the terminal window on startup
- You can also make all .py scripts execute with pythonw.exe, setting this through the usual facilities

1. Launch a command prompt.
2. Associate the correct file group with .py scripts:
   ```
   assoc .py=Python.File
   ```
3. Redirect all Python files to the new executable:
   ```
   ftype Python.File=C:\Path\to\pythonw.exe "%1" %*
   ```

Tokens in Python

Python tokens are the small units of the programming language. Python supports **5** types of Tokens.

1. **Keyword:-** These are the dedicated words that have special meanings and functions. Moreover, the compiler defines these words. Moreover, it does not allow users to use these words.
2. **Identifiers:-** Identifiers are the names given to any variable, function, class, list, method, etc. for their identification. Python is a case-sensitive language, and it has some rules and regulations to name an identifier.
3. **Literals:-** Literals are the fixed values or data items used in a source code. Python supports different types of literal.
   - **String Literals:** The text written in single, double, or triple quotes represents the string literals in Python.
   - **Numeric Literals:** These are the literals written in form of numbers.
   - **Boolean Literals:** Boolean literals have only two values in Python. These are True and False.
   - **Special Literals:** Python has a special literal 'None'. It is used to denote nothing, no values, or the absence of value.
4. **Operators:-** These are the tokens responsible to operate in an expression. The variables on which the operation is applied are called *operands.* Operators can be unary or binary. Unary operators are the ones acting on a single operand like a complement operator, etc. While binary operators need two operands to operate.
5. **Punctuators:-** These are the symbols used in Python to organize the structures, statements, and expressions. Some of the Punctuators are: [ ] { } ( ) @ -= += *= //= **== = , etc.

Using Variables

Variable Assignment

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Rules for Variable Names

- names can not start with a number
- names can not contain spaces, use _ instead
- names can not contain any of these symbols
  :'",<>/?|\!@#%^&*~-+
- it's considered best practice that names are lowercase with underscores
- avoid using Python built-in keywords like list and str
- avoid using the single characters l (lowercase letter el), O (uppercase letter oh) and I (uppercase letter eye) as they can be confused with 1 and 0

Python Keywords

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

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

All the keywords except True, False and None are in lowercase and they must be written as it is.

Using Variables

Dynamic Typing

Python uses dynamic typing, meaning we can reassign variables to different data types. This makes Python very flexible in assigning data types; it differs from other languages that are statically typed (meaning once the variable type is set, it cannot be changed).

```
In [1]: my_dogs = 2

In [2]: my_dogs
Out[2]: 2

In [3]: my_dogs = ['Sammy', 'Frankie']

In [4]: my_dogs
Out[4]: ['Sammy', 'Frankie']
```

Pros:

very easy to work with
faster development time
reduce clutter and duplication/ repetition in code

Cons:

- may result in unexpected bugs!
- you need to be aware of type()
- more errors detected later in development and in maintenance
- more errors at runtime and in shipped code.
- tends to prohibit compilation and yields poor performing code.
- need to write entirely mechanical tests for type correctness.
- hard to use to build a typed domain language.

Using Variables

Assigning Variables

Variable assignment follows name = object, where a single equals sign = is an assignment operator (which assigns the value of its right operand to its left operand)

```
In [5]: a = 5

In [6]: a
Out[6]: 5
```

Here we assigned the integer object 5 to the variable name a. Let's assign a to something else:

```
In [7]: a = 10

In [8]: a
Out[8]: 10
```

We can now use a in place of the number 10:

```
In [9]: a + a
Out[9]: 20
```

Reassigning Variables

As the word variable implies, Python variables can be readily changed. This means that you can connect a different value with a previously assigned variable very easily through simple reassignment.

Being able to reassign is useful because throughout the course of a program, you may need to accept user-generated values into already initialized variables or may have to change the assignment to something you previously defined.

Knowing that you can readily and easily reassign a variable can also be useful in situations where you may be working on a large program that was begun by someone else and you are not clear yet on what has already been defined.

Using Variables

Python let us reassign variables with a reference to the same object.

```
In [10]: a = a + 10

In [11]: a
Out[11]: 20
```

There's actually a shortcut for this. Python let us add, subtract, multiply and divide numbers with reassignment using +=, -=, *=, and /=.

```
In [12]: a += 10

In [13]: a
Out[13]: 30

In [14]: a *= 2

In [15]: a
Out[15]: 60
```

Determining variable type with type()

We can check what type of object is assigned to a variable using Python's built-in type() function. Common data types include:

- int (for integer)
- Float
- str (for string)
- List
- Tuple
- dict (for dictionary)
- Set
- bool (for Boolean True/False)

```
In [16]: type(a)
Out[16]: int

In [17]: a = (1,2)

In [18]: type(a)
Out[18]: tuple
```

Simple Exercise

This shows how variables make calculations more readable and easier to follow.

```
In [19]: my_income = 100
         tax_rate = 0.1
         my_taxes = my_income * tax_rate

In [20]: my_taxes
Out[20]: 10.0
```