1. **The paradigm of the language**

Since Python is built as a general purpose language, its paradigm falls into several categories:

1. Object-oriented
2. Imperative
3. Functional
4. Procedural
5. Reflection
6. **Some historical account of the evolution of the language and its antecedents**

Python was founded by Guido can Rossum at Centrum Wiskunde & Informatica (CWI) in Netherlands in the late 1980s, originally as an object-oriented language that also supported procedural programming (version 0.9.0). Successor to ABC language, it was designed to appeal to Unix/C hackers. Implementation began December 1989, and, from contributions of others, it reached version 1.0, including features like lambda, map, filter, and reduce.

With the core development team moving to BeOpen.com to form PythonLabs team, Python 2.0, released October 16, 2000, included many new features (cycle-detecting garbage collector, support for Unicode, among others). Development became more transparent and community-backed, and Python found popularity through its clean coding capability and overall learnability due to simple syntax.

Python 3.0 (initially called “Python 3000” or “py3k”) was released December 3, 2008 after a long testing period. Changes were made to many aspects (print statement, libraries, syntax), and new ones were added (bin function, argument handling, chained exceptions). Since the revisions were so substantial, it is not backward-compatible with previous version of Python. Many of its features have been backported () for Python 2.6.x and 2.7.x version series.

Python 2.7’s end-of-life date has been postponed from 2015 to 2020 due to concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler. The Register speculated that this was in response to Python 2.7's planned end-of-life, but Google cited performance under concurrent workloads as their only motivation.

Python 3.6 had changes regarding UTF-8 (in Windows, PEP 528 and PEP 529), and Python 3.7.0b1 (PEP 540) adds a new "UTF-8 Mode" (and overrides POSIX locale).

**Antecedents**

* 1. ABC
  2. ALGOL
  3. C
  4. C++
  5. CLU
  6. Dylan
  7. Haskell
  8. Icon
  9. Java
  10. Lisp
  11. Modula-3
  12. Perl

1. **The elements of the language: reserved words, primitive data types, structured types**

**Reserved words:**

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

**Primitive data types:**

**Structured types:**

|  |  |
| --- | --- |
| * list * tuple | * dictionary * set |

* Boolean
* Numeric: int, long, float, complex

1. **A description (in some form) of the syntax of the language**

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).

A python program contains logical lines, which consist of at least one physical line. A logical line can only be terminated by a newline token. *A particularly useful example shows the flexibility of the interpreter (Figure 1)*

Lines that contain only whitespace along with an optional comment are considered blank lines and are ignored by the interpreter. Python supports solely single-line comments, which begin with a hash character (#).

A logical line can be broken into multiple physical lines with the backslash character (\) but values cannot be split between lines (As a note, Python also allows strings to be broken up into multiple lines (preserving newline characters) with the use of triple quotes (""") surrounding the text.). Also, multiple logical lines can be included in one physical line with the use of a semicolon (;) between each statement.

Blocks are defined with whitespace (spaces and tabs) rather than the common counterpart, braces ({}). Usability is facilitated by allowing arbitrary yet consistent spacing.

1. **The basic control abstractions of the language (loops, conditional controls, etc.)**
2. if/elif/else statements
3. while/for loops (in range, in elements of structure)
4. iterators (
5. list comprehension expressions *(enhanced for loop?)*
6. break/continue/pass statements
7. try/except/finally statements
8. **How the language handles abstraction (including functions, procedures, objects, modules, etc.)**

With Python’s focus on readability, many complex components (whether a function, object, or package) are drastically simplified, sometimes into a single line. Since user’s can define their own components, this abstraction can be broadened to further complex ideas.

1. **An evaluation of the language's writability, readability, and reliability using the criteria discussed in chapter and of the textbook.**

**Writability**

* 1. **Simplicity and Orthogonality:**
     1. Python’s design emphasizes explicitness, which facilitates familiarity and utilization. A common standard in the language is to include a function within any type of unit abstraction that prints the documentation (or guide) to explain its functionality, usually including a comprehensive list of functions and attributes.
  2. **Support for abstraction:**
     1. With the ease and speed of defining and calling a function, abstraction in this manner is supported heavily. There are also many predefined functions, some of which encapsulate complex methods.
     2. Python practices dynamic typing, which means upon assignment, a variable will determine its data type once the actual value is returned. This allows for generic typing to be done automatically, shortening code and quickening the logical process. Complex data structures also are designed as clearly as possible, and, if mutable, are easily accessed and manipulated.
  3. **Expressivity:**
     1. This is one of the main foundations of Python ideology. Being able to say more with less is hard to avoid in this language. This can be found in functions, data structures, modules, and packages. With such a high level of abstraction, intuitive expressions are increasingly efficient.

**Readability:**

* 1. **Simplicity:**
     1. While it can be difficult to differentiate its structured types, there is a relatively small amount of effort required to understand the most basics concepts of the language and how they are implemented. With such a high level of abstraction, simplicity naturally follows.
     2. Feature multiplicity is supported in Python, yet the language strives to provide a single, most-obvious way to accomplish a task. Still, users are given an open range of types and functions to use based on their preferences. Examples include building structured types, analyzing data, and presenting output.
     3. Python allows users to overload practically any operator by defining predetermined functions in an object’s class. If the user desires to add two circle objects together to produce a new circle, a function can be defined with the object’s class that executes this feature. It can be decided whether to add the perimeter lengths, area, etc.
  2. **Orthogonality:**
     1. This aspect is capitalized in Python, particularly with ­dynamic typing. There is complete overlap of constructs by allowing the variable to accept any data type. Functions, especially regarding data structures, also provide some overlap, yet there is some restriction when working with multiple data types within a single function. The level of orthogonality was set to provide both straightforwardness and logic.
  3. **Data types: comprehensive and understandable**
  4. **Syntax design**
     1. Special words (reserved)
     2. Form and meaning (obvious and logical)

1. **The major strengths and weaknesses of your language.**

The language's core philosophy is summarized by aphorisms, such as:

*“Beautiful is better than ugly*

*Explicit is better than implicit*

*Simple is better than complex*

*Complex is better than complicated*

*Readability counts”*

- Zen Python??

Even though Python has a large standard library, it was designed with high extensibility in mind. Furthermore, the syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++. With these reasons, it is clear that Python is determined to be a general purpose language.

Still, it has some drawbacks. A major disadvantage of the language is averagely longer testing periods, with most errors occurring at run-time. In certain situations, Python has shown slower performance than other languages executing the same functionality, yet higher performance in other cases is a common counter-argument. Contrastingly, Python developers have found no viable rebound to its weakness in mobile computing. With these drawbacks, it is left for the user to decide whether or not Python is the most suitable language for the task at hand.

1. **An overview of the programs that you included and a discussion of what language features they highlight and how the language made the programs easy/hard to implement.**

**If-Elif-Else Castle Journey**

This program highlights writability and readability. With the story prewritten, organizing and writing the nested conditional statements took about 10 minutes. A user-defined function is also implemented to handle incorrect input, which further enhances both abilities. Multiple people with no coding experience understood the logic flow and syntax of the program within a single minute.

A major complication of writing complex code is incorrectly placed, missing, or extra braces. Python mitigates that immediately by only requiring indentation, and similar benefits can occur regarding no requirement of semicolons at the end of logic lines.

**Text Statistics**

This program highlights the ease with forming, decomposing, and manipulating data structures, all done with less code compared to Java.

A dictionary was used to store individual characters in a text file and their total frequency. The dictionary was then converted into a tuple then sorted in two different ways, resulting in a total of three tuples. Each were printed by simply passing the variable through a print statement, and the output was automatically organized and easily readable without further manipulation.

Python efficiently and conveniently provides mechanisms to work with data structures in any way, increasing writability, readability, and reliability.

<https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121>

<https://en.wikipedia.org/wiki/Python_(programming_language)>

<https://www.protechtraining.com/content/python_fundamentals_tutorial-functions>

<https://docs.python.org/3/>

<https://www.safaribooksonline.com/library/view/python-in-a/0596001886/ch04s09.html>

<https://python.swaroopch.com/control_flow.html>