

Wave

A Markup Language written in Python which transpiles JSON to HTML with inline CSS.

Project Report on:

“Wave: A Markup Language that transpiles JSON to HTML with inline CSS”

Submitted to Central Board of Secondary Education, Delhi

In partial fulfillment of the requirement of XII

(Computer Science)

Examination

2020-21

Guide: Vaibhav D. Bhusari

Submitted By: Shreyas Sable

Computer Science

Alphonsa Sr. Sec. School, Swangi Meghe, Wardha

2020-21



Contents

Topic	Page No.
Certificate	03
Acknowledgement	04
Distribution	05
Modules Used	06
Requirements	07
Source Code	08
Explanation & Output	14
Limitations	36
Bibliography	37

Certificate

This is to certify that _____ of class XII,
Alphonsa Senior Secondary School, Sawangi Meghe, Wardha has successfully
completed his project in Computer Practical (Python) as prescribed by CBSE in
the year 2020-21.

Date:

Place:

Signature of Principal

Signature of Internal Examiner

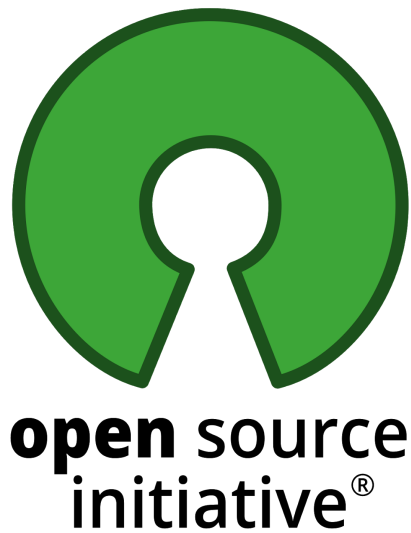
Signature of External Examiner

Acknowledgement

I thank my Computer Science teacher Mr. Vaibhav Bhusari for guidance and support. I also thank my Principal Rev. Sr. Rosanto. I would also like to thank my parents for encouraging me during the course of this project. Finally, I would like to thank CBSE for giving me this opportunity to undertake this project.

Signature of Student

Distribution



Wave is entirely free and open-source and is available on GitHub. Wave is distributed under MIT License.

Wave

<https://www.github.com/Waveryder2020/Wave>

Modules Used

sys

for accessing command-line arguments using **sys.argv**.

pathlib

for checking if a path exists or not using **pathlib.Path().exists()**.

json

for converting a JSON format file into a Python Dictionary using **json.load()**.

Requirements

Hardware Requirements

Processor: Intel i3 or greater. (ARM and AMD works too).

RAM: 1 GB or more.

Hard-Disk: 500 GB or more.

Software Requirements

Operating System: Windows / Linux / macOS

Python 3 Interpreter: For running Wave Transpiler.

A text editor: For writing programs in Wave.

Document Viewer: For reading documentation.

Source Code

The following is the source code of the project which is the Wave language transpiler. The entire language transpiler is made up of only one file, **wave.py**.

```
#!/usr/bin/env python3

import sys
import pathlib
import json

sp = " " * 4
html_body = ""

p_title = "Wave Document"
p_bgcolor = "white"
p_bgimage = "none"
p_align = "left"
p_box = 0
p_box_style = "hidden"

def key_exists(dict_key, dictionary):
    keys = list(dictionary.keys())
    return (dict_key in keys)

def starts_with(long_str, sub_str):
    return (sub_str == long_str[0:len(sub_str)])

if len(sys.argv) == 1:
    path = input("Specify the path to the script: ")
else:
    path = sys.argv[1]
```

```
if pathlib.Path(path).exists():
    script_file = open(path, "r", encoding = "utf-8")
else:
    print(f"Invalid Path: {path}")
    exit()

script = json.load(script_file)
script_file.close()

if key_exists("~page", script):
    page_property = script["~page"]

    if key_exists("~title", page_property):
        p_title = page_property["~title"]
    if key_exists("~bg", page_property):
        p_bgcolor = page_property["~bg"]
    if key_exists("~pic", page_property):
        p_bgimage = page_property["~pic"]
    if key_exists("~align", page_property):
        p_align = page_property["~align"]
    if key_exists("~box", page_property):
        p_box = page_property["~box"]
    if key_exists("~box-style", page_property):
        p_box_style = page_property["~box-style"]
```

```
c_p_bgcolor = p_bgcolor
c_p_align = p_align
c_p_color = "black"
c_p_size = 17
c_p_box = 0
c_p_body = ""
c_p_points_type = "disc"
c_p_point_start = "ul"

def set_defaults():
    global p_bg_color, p_align, c_p_bgcolor, c_p_align, c_p_color, \
        c_p_size, c_p_box, c_p_body, c_p_points_type, c_p_point_start
    c_p_bgcolor = p_bgcolor
    c_p_align = p_align
    c_p_color = "black"
    c_p_size = 17
    c_p_box = 0
    c_p_body = ""
    c_p_points_type = "disc"
    c_p_point_start = "ul"
```

```

if key_exists("$content", script):
    content_property = script["$content"]

    if key_exists("$heading", content_property):
        heading = content_property["$heading"]
        html_body += ("\t<br>\n\t<h1 style = 'text-align: center;" + \
            f"font-family: Arial Narrow, sans-serif">{heading}</h1>\n")
    if key_exists("$author", content_property):
        author = content_property["$author"]
        html_body += "\t<br>\n\t<h2 style = 'text-align: center;" + \
            f"font-family: URW Chancery L, cursive"><i>{author}</i></h2>\n"

content_property_copy = content_property.copy()
if key_exists("$heading", content_property_copy):
    del content_property_copy["$heading"]
if key_exists("$author", content_property_copy):
    del content_property_copy["$author"]
regular_keywords = list(content_property_copy.keys())
regular_values = list(content_property_copy.values())

```

```

for keywords in range(0, len(regular_keywords)):
    if starts_with(regular_keywords[keywords], "!inherit"):
        inherit = content_property[regular_keywords[keywords]]

        if isinstance(inherit, str):
            if inherit == "!default":
                set_defaults()
            else:
                pass
        elif isinstance(inherit, dict):
            if key_exists("!size", inherit):
                c_p_size = inherit["!size"]
            if key_exists("!color", inherit):
                c_p_color = inherit["!color"]
            if key_exists("!align", inherit):
                c_p_align = inherit["!align"]
            if key_exists("!bg", inherit):
                c_p_bgcolor = inherit["!bg"]
            if key_exists("!box", inherit):
                c_p_box = inherit["!box"]
            if key_exists("!points-type", inherit):
                if inherit["!points-type"] in ["circle", "square", "disc"]:
                    c_p_point_start = "ul"
                    c_p_points_type = inherit["!points-type"]
                if inherit["!points-type"] in ["lower-roman", "upper-roman", "decimal", "lower-alpha", "upper-alpha"]:
                    c_p_point_start = "ol"
                    c_p_points_type = inherit["!points-type"]

    if starts_with(regular_keywords[keywords], "$text"):
        html_body += (f"\t<p style = 'color: {c_p_color}; background-color: {c_p_bgcolor}; font-size:" + \
            f"{c_p_size}px; text-align: {c_p_align}; margin: {c_p_box}px;'>{regular_values[keywords]}</p>\n")

    if starts_with(regular_keywords[keywords], "$points"):
        points = regular_values[keywords]
        points_head = (f"\t<{c_p_point_start} style = 'color: {c_p_color}; background-color: {c_p_bgcolor}; font-size:" + \
            f"{c_p_size}px; text-align: {c_p_align}; margin: {c_p_box}px; list-style-type: {c_p_points_type};'>\n")
        points_body = ""
        for c_p_points_join in range(0, len(points)):
            points_body += f"\t\t<li>{points[c_p_points_join]}</li>\n"
        points_complete = points_head + points_body + f"\t</{c_p_point_start}>\n"
        html_body += points_complete

    if starts_with(regular_keywords[keywords], "$nl"):
        times = int(regular_values[keywords])
        html_body += ("\n" + ("<br>" * times) + "\n")

    if starts_with(regular_keywords[keywords], "$pic"):
        html_body += (f"\t<img 'color: {c_p_color}; background-color: {c_p_bgcolor}; font-size:" + \
            f"{c_p_size}px; text-align: {c_p_align}; margin: {c_p_box}px;' src = '{regular_values[keywords]}'>")

```

```

html_top = f"""
<!--
This Document is generated using Wave.
Wave: https://www.github.com/Waveryder2020/Wave
-->

<!DOCTYPE html>
<html>
    <head>
        <title>{p_title}</title>
        <style>
            body {{
                background-color: {p_bgcolor};
                background-image: url({p_bgimage});
                text-align: {p_align};
                margin: {p_box}px;
                border-style: {p_box_style};
            }}
        </style>
    </head>
    """

html_document = html_top + f"\n{sp}<body>\n\n" + html_body + f"\n{sp}</body>\n</html>\n"

file_name = path.split(".")
if len(file_name) == 1:
    file_name.append(".html")
else:
    file_name[-1] = ".html"

out_name = "".join(file_name)
out_file = open(out_name, "w+", encoding = "utf-8")
out_file.write(html_document)
out_file.close()
print(f"Transpiled successfully to file: {out_name}")

```


Explanation & Output

Wave is a Markup Language which can be used for making documents, business cards, invitations, letters, etc.

Wave programs are written in JSON (JavaScript Object Notation) format as a simple text file. The text file can have any or no extension. The file is valid as long as it's a text file.

JSON format is a way of storing key-value pairs. It resembles a lot to a Python Dictionary. Wave uses special syntax which can be written in JSON format in a text file. This file is then passed to the Wave Transpiler (the **wave.py** file). The Wave Transpiler is written in Python 3. It takes the text file, converts it from JSON to a Python Dictionary, parses it and converts it to HTML and CSS. This HTML document file obtained is the desired output.

This documentation covers Wave on a Linux Distribution.

Goal

The goal of using Wave to make documents is to replace the use of heavy and resource consuming document making softwares like Microsoft Word. Wave shows that instead of using such softwares, you can just write your documents in a special format text file and then convert them into neat looking HTML documents.

Structure of a JSON file

A JSON file is simply a collection of key-value pairs. The keys are mostly strings while the values can be of types: Integers, Floats, Booleans, Strings, Lists and other JSON Objects.

A simple JSON file format:

```
{  
    "name": "Bruce Wayne",  
    "alias": "Batman"  
}
```

A JSON file starts with “{” and ends with “}”. The keys and the values are separated by a colon (:). All the key value pairs are separated from each other by a comma (,).

There should be no comma after the last key-value pair in a JSON Object.

Using the Wave Transpiler

The **wave.py** file is the Wave Transpiler. On running the file, it asks for a path to a script. This script refers to a Wave Program. Here, the user can specify the relative or absolute path to the Wave Program file.

Another way of specifying the Wave Program file is by using command-line arguments. The Wave Transpiler can take the path to the program file as a command-line argument.

Passing the file without using command-line argument:

```
| -> python3 wave.py  
Specify the path to the script: script.json  
Transpiled successfully to file: script.html
```

Passing the file using command-line argument:

```
| -> python3 wave.py script.json  
Transpiled successfully to file: script.html
```

Here, the program is named **script.json** with the **.json** extension. However this extension is not mandatory. The Wave Transpiler automatically generates the output HTML file with the same name as that of the program file but with the **.html** extension.

Structure of a Wave Program

A Wave Program is divided into two parts. These parts are known as “**Containers**”.

The first Container is called “**~page**”. This is where the properties related to the whole page are defined. All the properties inside the **~page** Container start with a tilde symbol (~).

Properties define the look of the page. Properties include title, background color (**~bg**), background image (**~bg**), alignment (**~align**), etc.

The second Container is called “**\$content**”. This is where the actual body of the document is defined. All the elements inside the **\$content** Container start with a dollar sign (\$).

Elements define the contents of the page. It includes **\$heading**, **\$author**, **\$text**, etc.

\$heading and **\$author** are reserved elements. They can be used only once in the entire program.

Basic Wave Program Structure:

```
{  
  “~page”: {  
    All page properties go here.  
  },  
  
  “$content”: {  
    Body of the page go here.  
  }  
}
```

There is also a **sub-container** called “**!inherit**” inside the **\$content** container. The **!inherit** sub-container is similar to the **~page** container, except unlike **~page**,



which applies its properties to the entire document, the **!inherit** sub-container applies its properties to all the elements that come after it.

In different words, all the elements that come after **!inherit** sub-container inherits all the properties from it. The properties for other elements can be overridden using another **!inherit** sub-container.

No reserved elements can be used more than once. Non-reserved elements can be used more than once. However, their names should not be the same. Any name is valid as long as it starts with that particular element name. For example: After using **\$text** once, the user can use the same element again, but with a different name like, **\$text-1**. This name is valid as it starts with the element name which is **\$text**.

Writing our first Wave Program

This is how our first and basic Wave Program looks like:

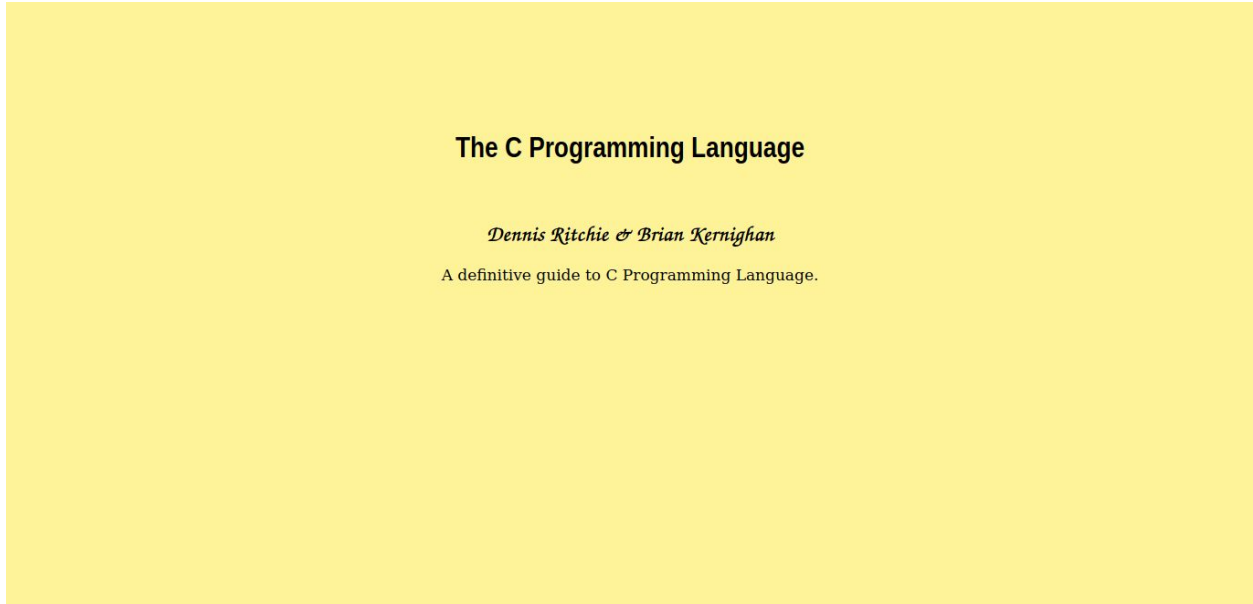
```
{
  "~page": {
    "~bg": "#fff39a",
    "~align": "center",
    "~box": "100"
  },

  "$content": {
    "$heading": "The C Programming Language",
    "$author": "Dennis Ritchie & Brian Kernighan",
    "$text": "A definitive guide to C Programming Language."
  }
}
```

Let's name this program "**script.json**". This program is to be transpiled from Wave (JSON format) to HTML + CSS. The Wave Transpiler is to be used for this.

```
| -> python3 wave.py
Specify the path to the script: script.json
Transpiled successfully to file: script.html
```

The HTML file, **script.html** is our desired output. The rendered document when opened in a web browser looks like follows:



All the properties and the elements will be discussed later in this document. For now, let's use the **!inherit** sub-container for adding properties to the **\$text** element.

```
{  
  "~page": {  
    "~bg": "#fff39a",  
    "~align": "center",  
    "~box": "100"  
  },  
  
  "$content": {  
    "$heading": "The C Programming Language",
```

```
"$author": "Dennis Ritchie & Brian Kernighan",

"!inherit": {
    "!color": "white",
    "!bg": "black",
    "!size": "50",
    "!align": "right",
    "!box": "50"
},

"$text": "A definitive guide to C Programming Language."
}
}
```

Here the same program has been modified. The **!inherit** sub-container modifies the properties of the **\$text** property.

The generated HTML document looks like this:

The C Programming Language

Dennis Ritchie & Brian Kernighan

A definitive guide to C Programming
Language.

Here is the table of the page properties which can be specified in the **~page** container.

Property	Description	Example
~title	Sets title in the browser.	“~title”: “Hello There”
~bg	Sets the background color. The user can specify the color either by name or by the color’s hex code.	“~bg”: “black”
~pic	Sets an image as the background image. The user can specify the absolute or relative path of the image.	“~pic”: “wallpaper.png”

~align	Aligns all the content of the document.	“~align”: “center”
~box	Sets a margin in the form of a box all around the document. The amount is specified in pixels.	“~box”: “100”
~box-style	Sets a style to the box.	“~box-style”: “dotted”

Here’s a list of all possible box styles:

hidden

dotted

dashed

solid

double

groove

rigid

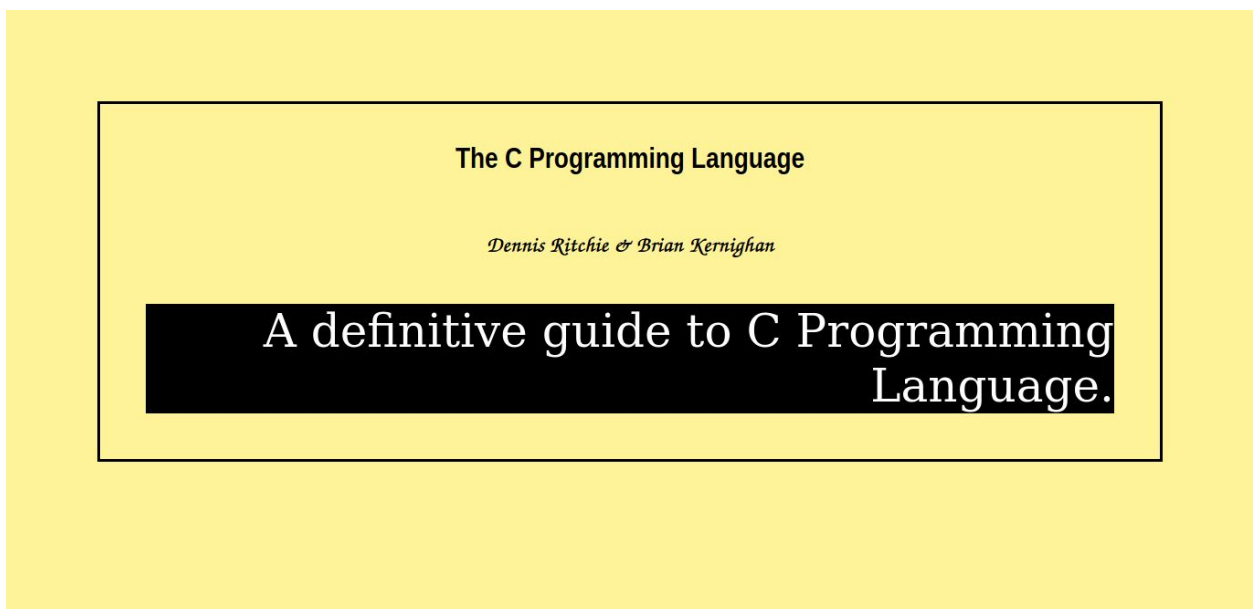
inset

outset

Adding box style can give a neat look to the document. For example, box style can be added to the document by adding just one line.


```
"~page": {  
    "~bg": "#fff39a",  
    "~align": "center",  
    "~box": "100",  
    "~box-style": "solid"  
}
```

Here **#fff39a** is the hex code for Dark Cream color which forms the background color of the document.



You'll have to re-transpile the Wave program every time you make changes and reload the page in the Web Browser for your changes to take effect in the document.

The \$content Container

As discussed earlier, the body of the document goes in the **\$content** container. Apart from the **!inherit** sub-container, the **\$content** container has the following elements:

\$text

\$points

\$pic

\$nl

Element	Description	Example
\$text	Inserts text into the document	“\$text”: “Hello Wave!”
\$points	Enters a list of points into the document. The user has to pass in the points as a list.	“\$points”: [“Markup Language”, “Transpiles to HTML”]
\$pic	Inserts an image in the document. Absolute or relative path of the image can be specified.	“\$pic”: “c_language.png”
\$nl	Adds new line(s). The user needs to specify how many new lines are to be	“\$nl”: 5

	added. The value of \$nl has to be passed as an integer and not as a string.	
--	---	--

The HTML tags, **** and **<i>** can be used inside the **\$text** element to make any part of the text bold and italic.

Just like the **!inherit** sub-container, other elements in the **\$content** container cannot have the same name. The elements are valid as long as their name starts with the element's name. Like, after using **\$text**, the second time you want to use text, you cannot directly write **\$text**. Instead, you can write **\$text-1** and so on.

Let's make this small change to our previous program and observe the changes in the HTML document.

```
"!inherit": {
    "!color": "white",
    "!bg": "black",
    "!size": "20",
    "!align": "right",
    "!box": "50"
},
```

```
"$text": "A definitive guide to <b>C Programming Language</b>.",
```

```

"!inherit-1": {
    "!color": "red",
    "!align": "left"
},

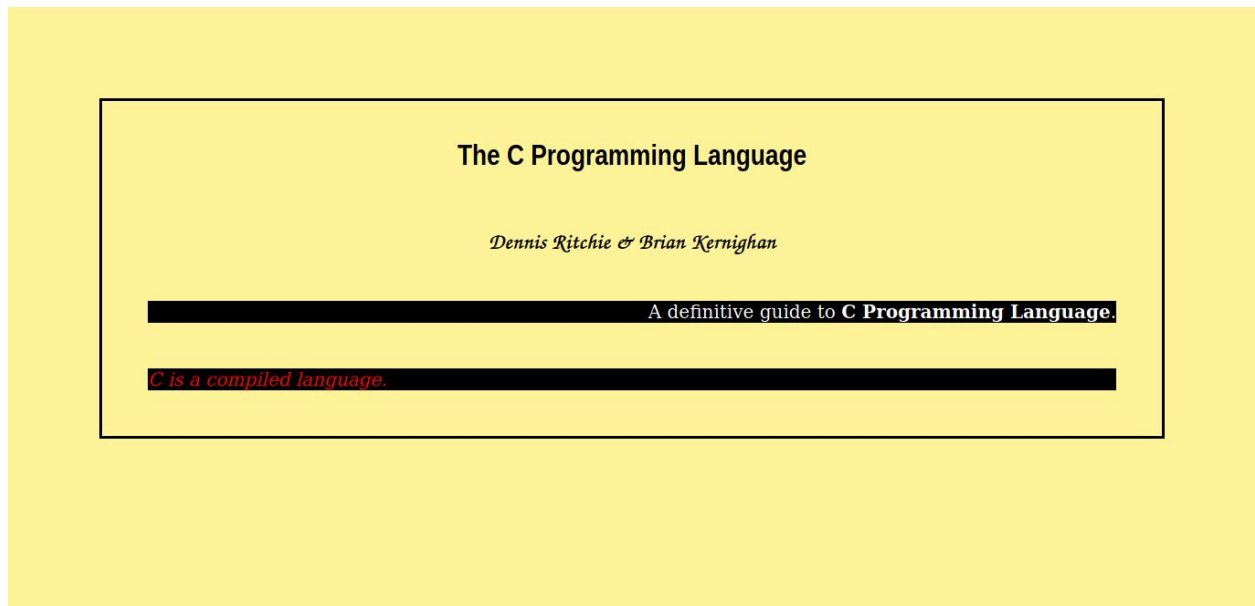
```

```

"$text-1": "<i>C is a compiled language.</i>"

```

Here the **!inherit-2** sub-container inherits the non specified properties from the first **!inherit** sub-container.



These properties can be set to default properties by setting the latest **!inherit** sub-container to **!default**. Like:

```

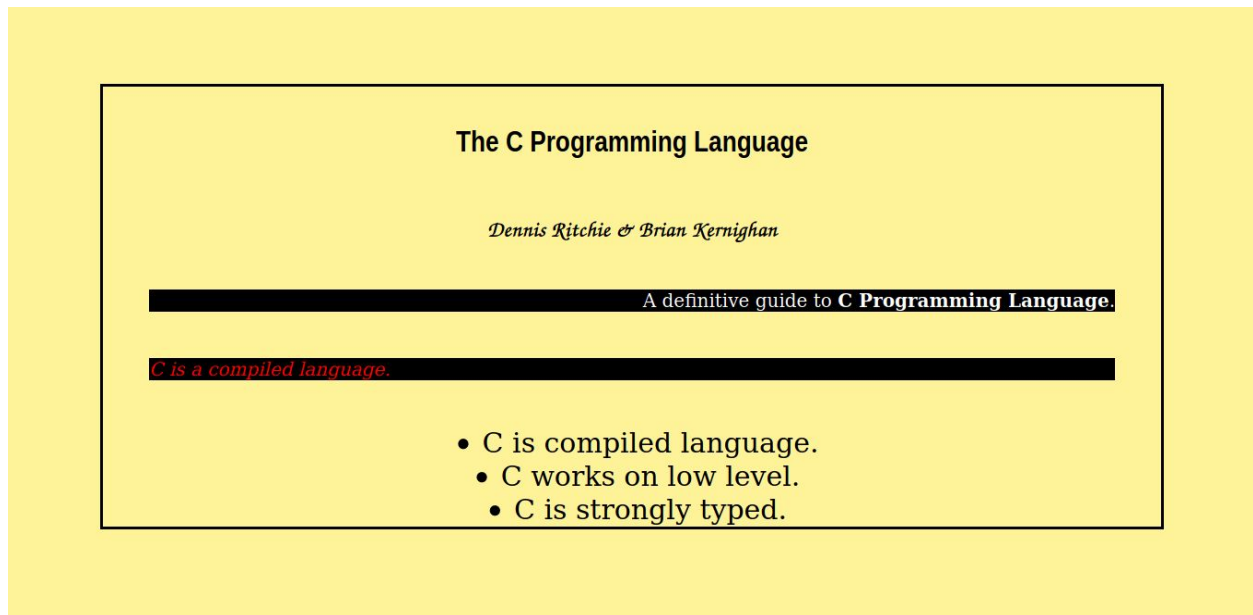
"!inherit-3": "!default"

```

Points can be added easily to a document. Points can be added by making few modifications to the previous program like:

```
"!inherit-3": "!default",  
"!inherit-4": {  
    "!size": "30"  
},  
  
"$points": [  
    "C is compiled language.",  
    "C works on low level.",  
    "C is strongly typed."  
]
```

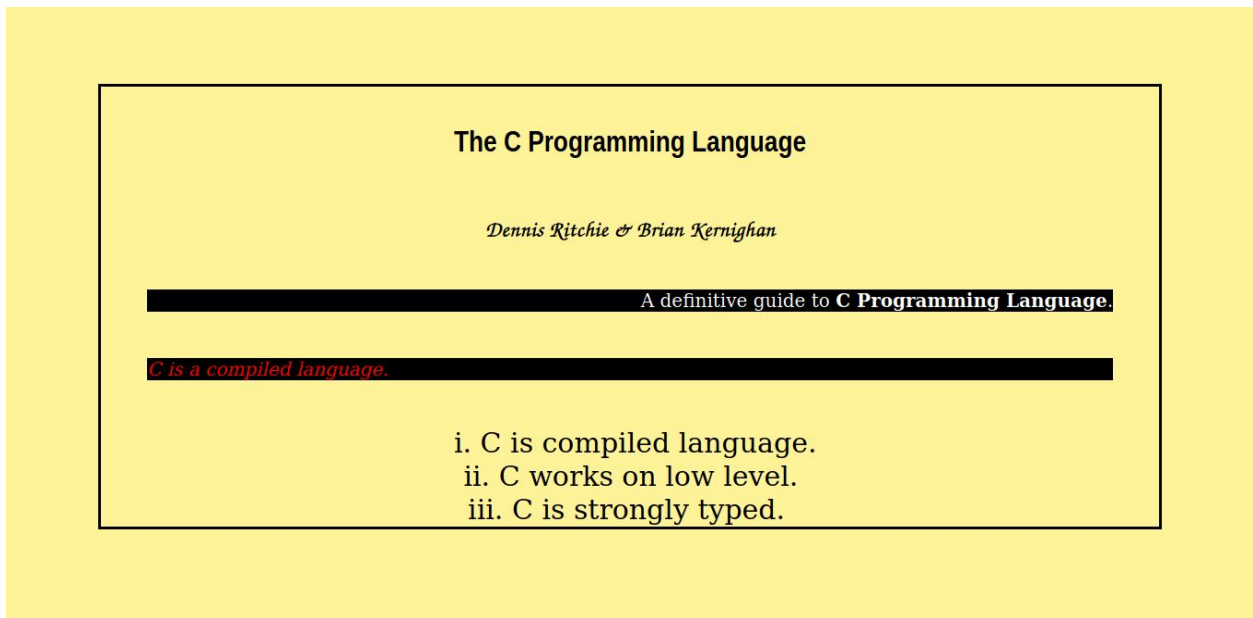
The rendered document on re-transpiling and reloading would look like this:



The type of points can also be set using the **!points-type** property in a **!inherit** sub-container. The previous program can be modified as follows:

```
"!inherit-4": {
  "!size": "30",
  "!points-type": "lower-roman"
},
```

The rendered document on re-transpiling and reloading would look like this:



Following is a list of all possible points types:

circle

square

disc

lower-roman


upper-roman

lower-alpha

upper-alpha

decimal

Images can also be added in the document using the **\$pic** element. The user needs to specify the address of the image.

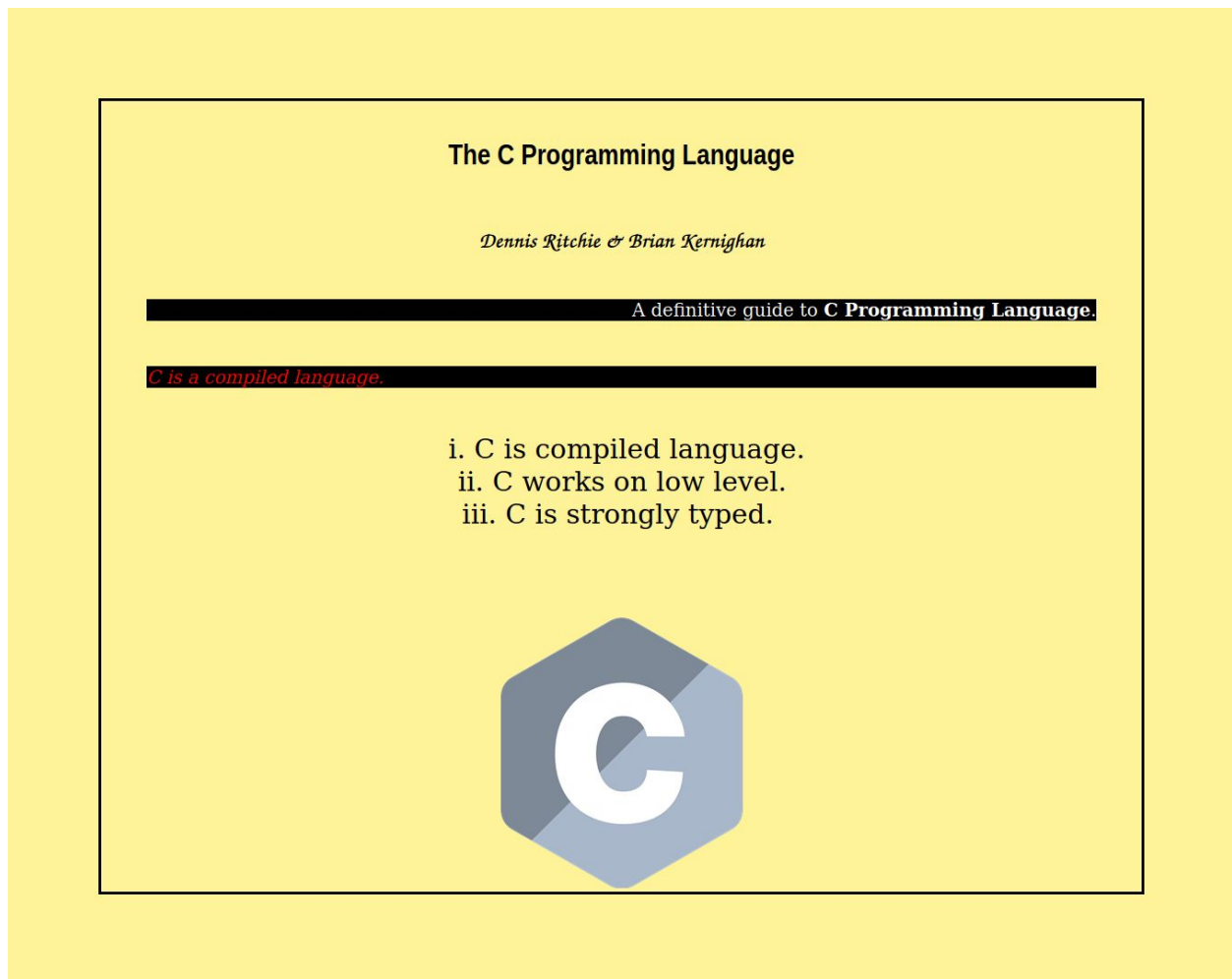


Consider this modification to our previous program for adding a few new lines and an image.

Here we have an image file **c_language.png** in the same directory as that of **wave.py** file and **script.json** file.

```
"$nl": 5,  
    "$pic": "c_language.png"
```

On re-transpiling, the document would look like this:



Here, this document has covered all the aspects of using the Wave Markup Language.

Complete Program and Output

1] Writing a Wave Program (**script.json**) (Complete Program):

```
{
  "~page": {
    "~bg": "#fff39a",
    "~align": "center",
    "~box": "100",
    "~box-style": "solid"
  },

  "$content": {
    "$heading": "The C Programming Language",
    "$author": "Dennis Ritchie & Brian Kernighan",

    "!inherit": {
      "!color": "white",
      "!bg": "black",
      "!size": "20",
      "!align": "right",
      "!box": "50"
    },

    "$text": "A definitive guide to <b>C Programming Language</b>.",
```

```
"!inherit-1": {
    "!color": "red",
    "!align": "left"
},

"$text-1": "<i>C is a compiled language.</i>",

"!inherit-3": "!default",
"!inherit-4": {
    "!size": "30",
    "!points-type": "lower-roman"
},

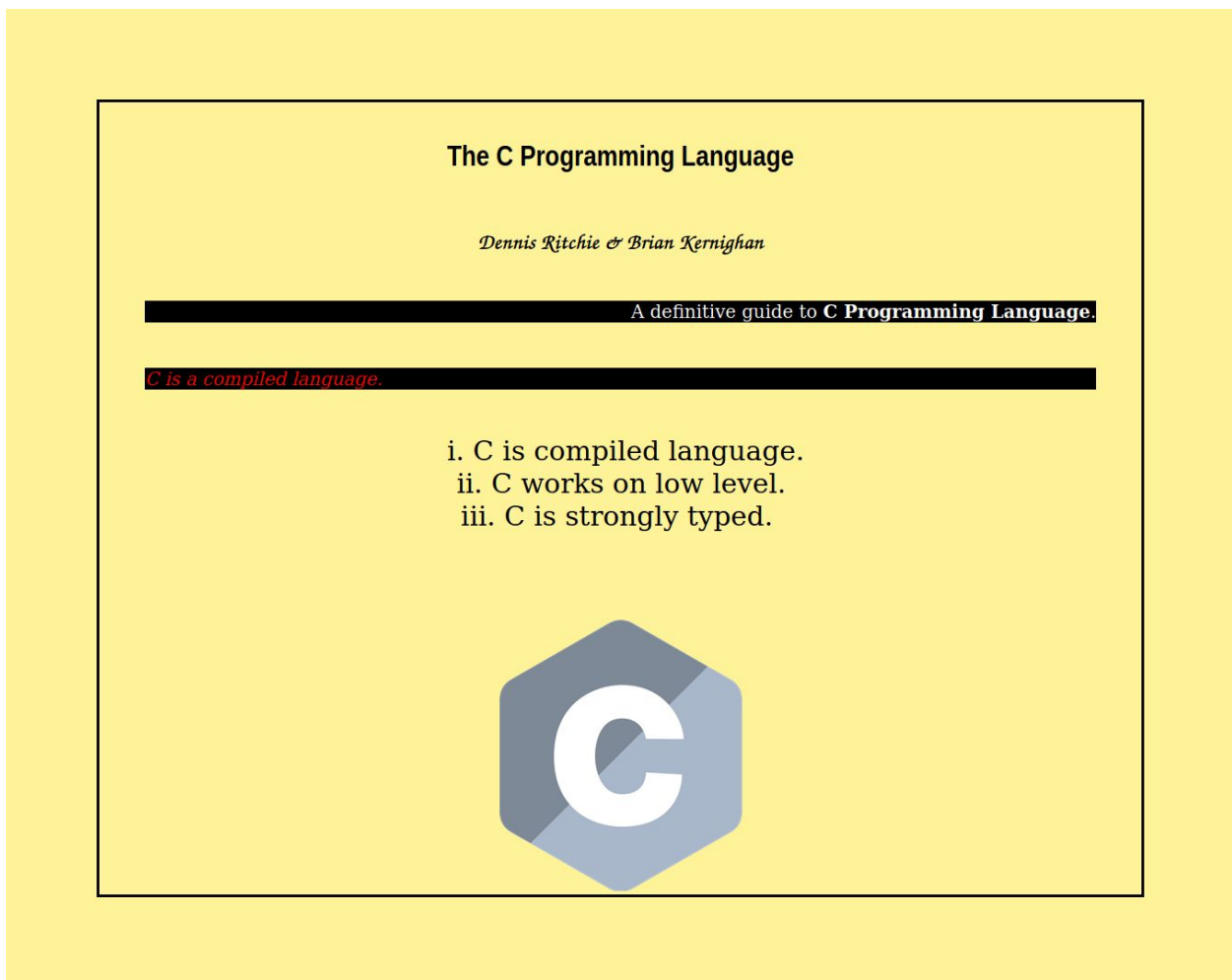
"$points": [
    "C is compiled language.",
    "C works on low level.",
    "C is strongly typed."
],

"$nl": 5,
"$pic": "c_language.png"
}
}
```

2] Transpiling the **script.json** program to HTML file, **script.html** using the Wave Transpiler (**wave.py**) which is written in Python 3:

```
| -> python3 wave.py  
Specify the path to the script: script.json  
Transpiled successfully to file: script.html
```

3] Opening the **script.html** file in a Web Browser:



Limitations

Wave has the following limitations:

1. Since Wave is a transpiled language, transpiling large Wave Programs can take a while.
2. Even if Wave transpiles to HTML & CSS, Wave programs do not have access to all the HTML elements and CSS properties.
3. Even if Wave rendered documents open in a Web Browser, Wave is suitable only for making documents like Letters, Invitations, Business Cards, etc. It's not suitable for making fully functional Web Pages.

Bibliography

The following resources were helpful for the creation of Wave and this document:

GeeksForGeeks

For taking a quick glance at a few functions.

(<https://www.geeksforgeeks.org>)

Google Docs

For making this document.

(<https://docs.google.com>)