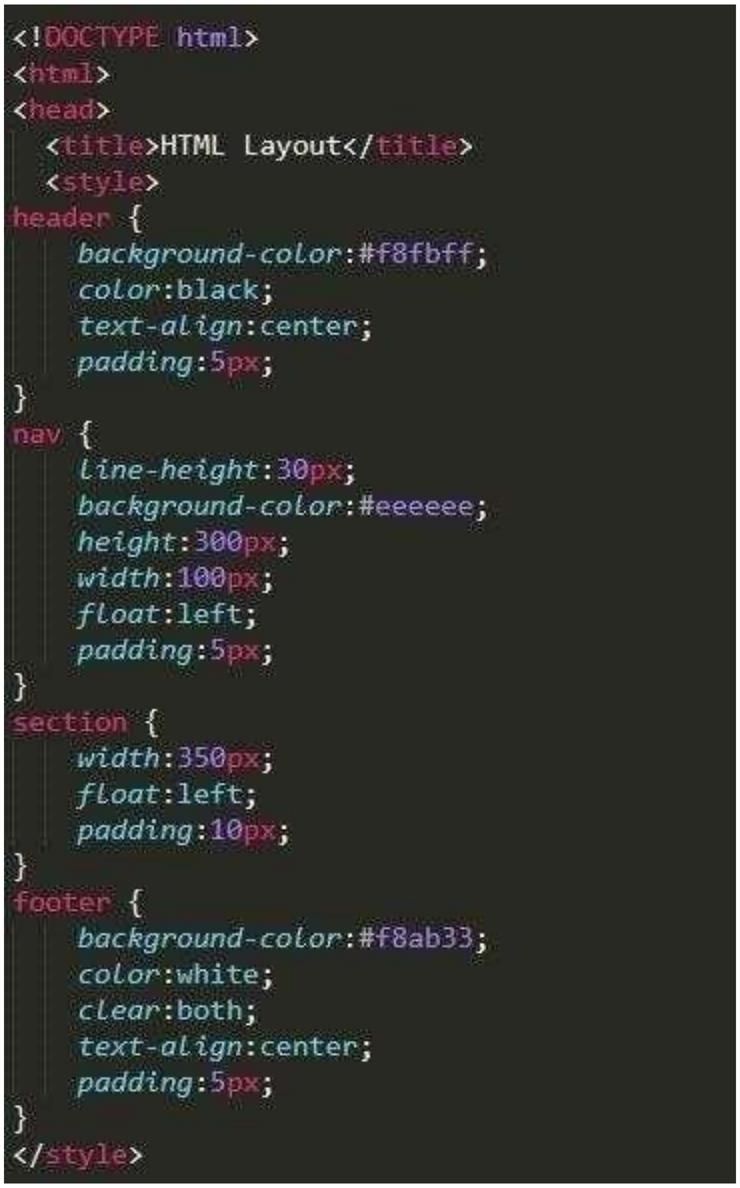
Date	**17 November 2022**
Team ID	**PNT2022TMID21423**
Project	**A Novel Method For Handwritten Digit Recognition**

Coding layout,readability ,reusability

Coding layout

Let’s understand the layout of an HTML web page using an example.



```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Layout</title>
  <style>
header {
    background-color:#f8fbff;
    color:black;
    text-align:center;
    padding:5px;
}
nav {
    line-height:30px;
    background-color:#eeeeee;
    height:300px;
    width:100px;
    float:left;
    padding:5px;
}
section {
    width:350px;
    float:left;
    padding:10px;
}
footer {
    background-color:#f8ab33;
    color:white;
    clear:both;
    text-align:center;
    padding:5px;
}
</style>
```

code readability

Readable code is simply code that clearly co mmunicates its intent to the reader. Most likely, the code we write will be read by other developers, who will eit her want to understand or modify the way our code works. They may need to test the code, fix a problem, or add a new feature.

These other developers reading our code could be our teammates, a consultant, a new junior developer, or even developers fro m another company who are programming a specific portion of the applicat ion. And more importantly, that developer could be ourselves, when we revisit our own code. With most languages, the ratio of time spent reading vs. writing is well over 10:1.

Why should we produce readable code?

Since our software code can be shared wit h other developers, we'll want to make it easier to work with this shared code. To do this, we'll have to make sure that everyone involved, including ourselves, understands the same thing easily and quickly. Code that is readable for one person is not necessarily readable for another.

Code that is not readable takes more time to understand and can make us lose a lot of time on what should be a simple task. The worst-case scenario is that it could even take several iterations to fix so me problems. In some cases, we might spend so much time trying to understand code that we might want to rewrite it completely.

Poorly written code would even make refactoring difficult. When rewrit ing poorly written code, by remo ving the original code and writ ing a new implementation, we might not cover all the use cases of that code, especially if the documentation of the requirements is limited or absent. As a result, the time spent rewriting will be greater than the time it took to write the original code.

Furthermore, by not understanding the code well, we may misinterpret its use and by modifying the code, we may unintent ionally create new problems. Code that is not easily readable can therefore increase the risk of defects.

Benefits of Creating Readable Code

Benefits of Creating Readable Code

On the other hand, readable and well-tested code is what makes it easier to refactor, extend and modify parts of the system because it is easier and less time-consuming to understand. Readable and well-tested code is the foundation of a solid base, where developers are confident and quick to make changes.

Good readable code obviously takes more effort to produce than quickly written and poorly planned code. At Direct Impact, we believe that the benefits of readable code win over quickly produced code that may contain more errors.

In order to produce good readable code, we will have to fo llow good coding practices.

How do we make our code more readable?

It would be easy to say that by simply adding comments to our code, it will automat ically beco me more readable. But readable code invo lves much more. Adding relevant comments everywhere in our code would be a very laborious task and could beco me overly co mmented code. Overly co mmented code would have the opposite effect, making the code harder to read and understand.

REUSABILITY

abstract

This paper presents a knowledge-based approach for select ing and testing modular reusable code. This approach entails three stages: system definit io n through a system ent it y structure (SES), SES pruning and model synt hesis using an expert system (ES), and the evaluat ion of candidate design models using discrete event simulat ion (DEVS). An example of this approach is shown through the development of a Group Decision Support System (GDSS) idea generation tool.

```
1:

                              Imme ifiate lease
    else
          reject Immediate lease and print message
else
    if szhe dufing te'ase:iype= =:BE then:
          Neue BBless e:md.'act stete ct lease. to gang.

          irt cheduling fete typ e= AH the
                if TiB late rsqqs st.      ady sc£e doled:ai ñi'aGfims then:
                   f mums «r of sump min»: > ono .limit. pro i'di d by u'ssr in
                   advance for.ihat'43B then:
                        Do not schedule this AR & reschedule BE to be
                        suspended on slot vacated by AR

                        Increase: c nunter. ñf s»sgension nf that KE and
                        aDoc ate resourced. to AR.lease
                iilse
                 if re9'ñuroes . are not iivnilqble a9 iiñirr'g. u8ed by odier' 'AR.
                 lease request then:

                âg   dizngMaie   e-=DkSñen:.
                        flerk=(deaâFne-sEid•bme\}durñAdo
                        if slack < 1.1 then
                         reject learn and print mcs cage to exiend dBad lina nr

                         'find a single time slut hich, can satisfy. c omjfets
                         le ase within deadhrie
                         if new lease. is not sche'dniabfe .as. nbove then:
                              .find 'mulopie gtnfs.hiBii: ioge'dier. can 'satisfy
                              'this.conditin ns
                         if néii leaie"ñ nut 's chedniabfi• by arp• of the abévé
                         methn de the.

                           reject new lease & print message

                        jjrint message imalid lease' type
```