Name: Dharmit Shah FYIT

Roll No: 75

1. Explain Green Computing With its advantages

Ans:- INTRODUCTION.

Green computing sometimes also called Green Technology. In the green computing we use computer and its related other resources such as monitor, printer, hard disk, floppy disk, networking in very efficiently manner which has less impact on the environment. Green computing is about eco-friendly use of computer. Green computing is important for all type of system. It is important for handheld system to large scale data centre[1]. Many IT companies have been start the use of green computing to reduce the environment impact of their IT operations[2].Green computing is the emerging practice of using computing and information technology resources more efficiently while maintaining or improving overall performance. The concept identifies the barriers and benefits of green computing

1.Decreased vitality use from green figuring systems converts into bring down carbon dioxide discharges, coming from a decrease in the non-renewable energy source utilized in control plants and transportation.

2. Saving assets implies less vitality is required to create, utilize, and discard items.

3. Sparing vitality and assets sets aside extra cash

4. Green is empower that recycling and taking down vitality use by and organizations people.

5. Diminish the hazard existing in the workstations, for example, substance known to cause disease, nerve harm and resistant responses in people.

2. **What is E-waste? What can be done to reduce the impact of E-waste.**
   Ans:-

Electronic waste, or e-waste, refers to all items of electrical and electronic equipment (EEE) and its parts that have been discarded by its owner as waste without the intent of re-use (Step Initiative 2014). E-waste is also referred to as WEEE (Waste Electrical and Electronic Equipment), electronic waste or e-scrap in different regions and under different circumstances in the world. It includes a wide range of products – almost any household or business item with circuitry or electrical components with power or battery supply. In this methodology, defined by the Partnership on Measuring ICT for Development (Baldé et al., 2015a), the definition of e-waste is very broad. It covers six waste categories:

1. Temperature exchange equipment, more commonly referred to as cooling and freezing equipment. Typical equipment includes refrigerators, freezers, air conditioners, heat pumps.

2. Screens, monitors. Typical equipment includes televisions, monitors, laptops, notebooks, and tablets.

3. Lamps. Typical equipment includes fluorescent lamps, high intensity discharge lamps, and LED lamps.

4. Large equipment. Typical equipment includes washing machines, clothes dryers, dish-washing machines, electric stoves, large printing machines, copying equipment, and photovoltaic panels.

5. Small equipment. Typical equipment includes vacuum cleaners, microwaves, ventilation equipment, toasters, electric kettles, electric shavers, scales, calculators, radio sets, video cameras, electrical and electronic toys, small electrical and electronic tools, small medical devices, small monitoring and control instruments.

6. Small IT and telecommunication equipment. Typical equipment includes mobile phones, Global Positioning Systems (GPS), pocket calculators, routers, personal computers, printers, telephones. Each product of the six e-waste categories has a different lifetime profile, which means that each category has different waste quantities, economic values, as well as potential environmental and health impacts, if recycled inappropriately. Consequently, the collection and logistical processes and recycling technology differ for each category, in the

same way as the consumers' attitudes when disposing of the electrical and electronic equipment also vary

3. What are the benefits of going paperless.

Ans:-

10 benefits of going paperless

The market is competitive for businesses. The pressure to improve efficiency and cut costs while still maintaining a high level of customer service is felt constantly. One great way to accomplish this is by going paperless.

Technology has made going paperless easier than ever, yet many companies still haven't made the switch. Many are unaware of the incredible benefits that paper-free processes can provide; others are intimidated at the thought of switching to entirely new processes. However, there's no need to feel overwhelmed; the benefits of going paperless are substantial, and it's easier than ever to remove the paper from your workflows and processes.

Here are 10 reasons your company should strongly consider going paperless.

1. Better organization
Manually organizing and keeping track of papers is time-consuming; a cluttered, messy stack of loose documents makes it take that much longer, especially if you're trying to find something specific. Disorganization can seriously hamper how much you can get done; the average person spends about **10 minutes per day** looking for lost items. That means that about **six months of your life** is spent looking for missing items.
Want to take back your time and redirect it to what matters? Start by opting for increased organization and streamlined processing for your paperwork. Going paperless will save you the headache of keeping up with the mess. You will also never have to fret about losing that *one* invoice that you need to get approved because it slipped through the cracks.

2. Collaboration
Digitization of your paper-based processes means that your whole team has the necessary access to information at all times. A file left on someone's desk, a random note with important information, a filing system that only one person understands… These can spell trouble for your business.

If a crucial person calls in sick, or you need to be away from the office, your employees may struggle to find the information that they need. Paper-based processes can cause a loss of productivity and a potential drop in customer service.

You can avoid a breakdown in necessary information-sharing by going paperless. Files, instead of potentially becoming lost, become accessible from anywhere at any time, allowing your team easier access to the information that they need.

3. An office environment that sparks joy

When guests visit your office, what do you inadvertently let them know about your company? Whether you mean to or not, the appearance of your office tells your customers and employees a lot about your business.

A workspace filled with stacks of papers is a thing of the past. An uncluttered, paper-free office gives a more professional, modern appearance for your customers; it also provides a more relaxing environment for your employees to work.

This type of streamlined organizational setting has an economic benefit as well. As most business owners know, office space isn't cheap. Instead of wasting office space storing paperwork, digitized systems hold all the same information in a fraction of the area. You can use your workspace for work instead of as a glorified storage unit.

4. Increased efficiency

Efficiency is at the heart of any successful business. When you minimize time required to be spent on busywork, your team can repurpose that time to focus on critical, value-add tasks. Going paperless means less time spent on clerical work and more time doing what matters.

Completing, filing, organizing, and keeping track of paperwork can create a severe strain on your time; time spent on paperwork is cut down to a minimum after going paperless.

Filling out paperwork is also simpler with technology; digitally-captured data can be used to create rules to automatically generate applications. Automated systems allow you to not only fill out information quickly but reliably as well, as validation algorithms reduce errors and ensure complete data.

5. Reduced costs

Your organization saves more than just paper by adopting a paperless office. The money spent on printer upkeep, ink, toner, postage, physical storage, to name a few, will all significantly decrease as your reliance on paper diminishes. This is why companies save an average of **$80 per employee** when they make the switch to a paperless system.

Your company will save more than just the physical costs of printing and paper. Increasingly efficient offices will repurpose your spending as well. For example, you pay for your employees' time; the more that they spend their time doing value-add work, instead of manual paperwork, the more return you'll see on the investment in them.

6. Enhanced security

Physical paper can be a significant liability for your company. Sensitive files can easily be compromised, and paperwork can be misfiled, destroyed, or stolen. Your customers' trust in you should depend on more than a locked filing cabinet and shredder.

A paperless approach offers a much higher level of security for your and your customers' sensitive data. Numerous safeguards, encryption, and banking-level security measure work together to protect your documents. In a paper-based office, everyone has access to all the information. In a paperless office, it is possible to give each employee only the specific access they require. For example, a medical office can limit the information a front desk receptionist can access but gives doctors complete access to medical history.

Cloud-based technology also provides back-up in case of an emergency. If your office is subject to a natural disaster, such as a flood or fire, you won't lose any vital information.

7. Easier access

A paperless system allows you to access your information from anywhere at any time. Whether you're in discussion with your employee at the office or traveling out of town, cloud-based storage means that you maintain access to your necessary information. It also improves customer experience when you can answer questions by quickly accessing necessary information.

Customers also appreciate personalization. A targeted campaign, using relevant metrics and based on their needs, will be much more effective for lead generation, up-sell, and retention. For example, a targeted email campaign will be easier to track, analyze, and optimize, as opposed to mailing out general promotions.

8. Compliance with regulations

Businesses are liable for the information that they keep on their customers. Regulations are in place to make sure that all information is kept secure from a potential breach. HIPPA and SOX regulations in America and GDPR in Europe all demand that companies save data with integrity. These regulations help uphold customer confidentiality and trust.

Complying with these regulations in a paper-based system is complicated. An open file on a desk, a customer that wanders into a confidential area, or a misplaced paper all hold risk of a potential violation and headache for you. Keeping sensitive information secure and up to compliance standards is much easier with built-in electronic protections and safeguards. You can easily create an audit trail to track your documents whenever you need.

9. Eco-friendly
Reducing your reliance on paper allows you to create a more sustainable business. Even with recycling, the amount of paper used in U.S. offices grows **20% a year**. The average employee uses an astounding **10,000 sheets of copy paper** per year. Your company can contribute to the well-being of the environment by cutting down on the amount of paper you use.
Beyond paper, the printer, ink, and ink cartridges are harmful to the environment as well. It takes over **3 quarts of oil** to create a laser printer and inkjet cartridges. Printers and ink also contain potentially harmful chemicals that, if disposed of improperly, pollute water and soil and contribute to the spread of ecological damage.

Not only is an eco-friendly approach more sustainable and better for the planet, it also helps your brand image. In today’s culture, most customers know how much waste is produced in an office and are concerned about the environment. You can showcase your company’s concern for the environment and how your business is making a difference by participating in different reporting initiatives and certification programs, such as ENERGY STAR and Fitwel.

10. Peace of mind
Working in a fast-paced environment is exciting; it can also be very stressful. You can reduce some of the stress by going paperless. The decreased clutter, seamless access to data and other assets, increased security, and reduced busywork all make for a more peaceful workspace.

You can reduce the chaos of paperwork, the fear of losing something important, eliminate errors, and strengthen the security of your organization. You can’t get rid of all the stress of running a business, but you can make it much easier.

Create a better office by going paperless
Switching to paperless doesn’t have to be intimidating; you don’t have to implement changes all at once. Even small reductions in day-to-day paper usage will start to uncover some cost savings. Start your journey to go paperless today and see how much it can benefit you!

4. What is Github ? Give advantages of using Github.

Ans:

Github is a hosting platform wherein developers can store their computer code in the github server in files and folders called repository and track them continuously. It is an open-source version control and collaboration

platform for program developers.
It helps all the programmers to collaborate with each other who are working on a similar project and also share their code easily as and when required. The collection of these files will shows the source code of a program which is spread across the files to make it easier to manage what can be many thousands of lines of code and still be able to find the parts you need to find.

Advantages of GitHub

It's free and it is open source:

As discussed earlier, github is completely free and you can use it without paying and since it is an open source you can download the source code and can make changes as per the requirements.

It is fast:

Since most of the operations are preferred locally, it allows huge benefit in terms of speed.

It provides good backup:

Here chance of losing data is very low as it provides the multiple copies of it.

Multiple developers can work:

Github allows multiple developers to work on a single project at a time. It helps all the team members to work together on a single project at at a time from different locations.

Conclusion

Once you know the usage of github repository you can go ahead and start using it. It has the best interface to its concepts and is a solid base to work from. Github is the most preferred platform of developers especially whenever the developers are working on a single project from different locations.

5. Write the program using PEP8 rules.

Ans:-

It gets difficult to understand a messed up handwriting, similarly an unreadable and unstructured code is not accepted by all. However, you can benefit as a programmer only when you can express better with your code. This is where PEP comes to the rescue.

Python Enhancement Proposal or PEP is a design document which provides information to the Python community and also describes new features and document aspects, such as style and design for Python.

Python is a multi-paradigm programming language which is easy to learn and has gained popularity in the fields of Data Science and Web Development over a few years and **PEP 8** is called the style code of Python. It was written by Guido van Rossum, Barry Warsaw, and Nick Coghlan in the year 2001. It focuses on enhancing Python's code readability and consistency. Join the certification course on Python Programming and gain skills and knowledge about various features of Python along with tips and tricks.

A Foolish Consistency is the Hobgoblin of Little Minds

'A great person does not have to think consistently from one day to the next' — this is what the statement means.

Consistency is what matters. It is considered as the style guide. You should maintain consistency within a project and mostly within a function or module.

However, there will be situations where you need to make use of your own judgement, where consistency isn't considered an option. You must know when you need to be inconsistent like for example when applying the guideline would make the code less readable or when the code needs to comply with the earlier versions of Python which the style guide doesn't recommend.

In simple terms, you cannot break the backward compatibility to follow with PEP.

The Zen of Python

It is a collection of 19 'guiding principles' which was originally written by Tim Peters in the year 1999. It guides the design of the Python Programming Language.

Python was developed with some goals in mind. You can see those when you type the following code and run it:

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

Namespaces are one honking great idea -- let's do more of those!

The Need for PEP 8

Readability is the key to good code. Writing good code is like an art form which acts as a subjective topic for different developers.

Readability is important in the sense that once you write a code, you need to remember what the code does and why you have written it. You might never write that code again, but you'll have to read that piece of code again and again while working in a project.

PEP 8 adds a logical meaning to your code by making sure your variables are named well, sufficient whitespaces are there or not and also by commenting well. If you're a beginner to the language, PEP 8 would make your coding experience more pleasant.

Following PEP 8 would also make your task easier if you're working as a professional developer. People who are unknown to you and have never seen how you style your code will be able to easily read and understand your code only if you follow and recognize a particular guideline where readability is your de facto.

And as Guido van Rossum said— "Code is read much more than it is often written".

The Code Layout

Your code layout has a huge impact on the readability of your code.

Indentation

The indentation level of line is computed by the leading spaces and tabs at the beginning of a line of logic. It influences the grouping of statements.

The rules of PEP 8 says to use 4 spaces per indentation level and also spaces should be preferred over tabs.

An example of code to show indentation:

```
x = 5
```

```
if x < 10:
  print('x is less than 10')
```

Tabs or Spaces?

Here the **print** statement is indented which informs Python to execute the statement only if the **if** statement is true. Indentation also helps Python to know what code it will execute during function calls and also when using classes.

PEP 8 recommends using 4 spaces to show indentation and tabs should only be used to maintain consistency in the code.

Python 3 forbids the mixing of spaces and tabs for indentation. You can either use tabs or spaces and you should maintain consistency while using Python 3. The errors are automatically displayed:

```
python hello.py
  File "hello.py", line 3
      print(i, j)
               ^
```

TabError: inconsistent use of tabs and spaces in indentation

However, if you’re working in Python 2, you can check the consistency by using a **-t** flag in your code which will display the warnings of inconsistencies with the use of spaces and tabs.

You can also use the **-tt** flag which will show the errors instead of warnings and also the location of inconsistencies in your code.

Maximum Line Length and Line Breaking

The Python Library is conservative and **79 characters** are the maximum required line limit as suggested by PEP 8. This helps to avoid line wrapping.

Since maintaining the limit to 79 characters isn’t always possible, so PEP 8 allows wrapping lines using Python’s implied line continuation with parentheses, brackets, and braces:

```
def function(argument_1, argument_2,
             argument_3, argument_4):
    return argument_1
```

Or by using backslashes to break lines:

```
with open('/path/to/some/file/you/want/to/read') as example_1, \
    open('/path/to/some/file/being/written', 'w') as example_2:
    file_2.write(file_1.read())
```

When it comes to binary operators, PEP 8 encourages to break lines before the binary operators. This accounts for more readable code.

Let us understand this by comparing two examples:

```
# Example 1

# Do

total = ( variable_1 + variable_2 - variable_3 )

# Example 2

# Don't

total = ( variable_1 + variable_2 - variable_3 )
```

In the first example, it is easily understood which variable is added or subtracted, since the operator is just next to the variable to which it is operated. However, in the second example, it is a little difficult to understand which variable is added or subtracted.

Indentation with Line Breaks

Indentation allows a user to differentiate between multiple lines of code and a single line of code that spans multiple lines. It enhances readability too.

The first style of indentation is to adjust the indented block with the delimiter:

```
def function(argument_one, argument_two,
             argument_three, argument_four):
    return argument_one
```

You can also improve readability by adding comments:

```
x = 10
if (x > 5 and
    x < 20):
    # If Both conditions are satisfied
    print(x)
```

Or by adding extra indentation:

```
x = 10
if (x > 5 and
      x < 20):
    print(x)
```

Another type of indentation is the **hanging indentation** by which you can symbolize a continuation of a line of code visually:

```
foo = long_function_name(
      variable_one, variable_two,
      variable_three, variable_four)
```

You can choose any of the methods of indentation, following line breaks, in situations where the 79 character line limit forces you to add line breaks in your code, which will ultimately improve the readability.

Closing Braces in Line Continuations

Closing the braces after line breaks can be easily forgotten, so it is important to put it somewhere where it makes good sense or it can be confusing for a reader.

One way provided by PEP 8 is to put the closing braces with the first white-space character of the last line:

```
my_list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
    ]
```

Or lining up under the first character of line that initiates the multi-line construct:

```
my_list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
```

Remember, you can use any of the two options keeping in mind the consistency of the code.

Blank Lines

Blank lines are also called vertical whitespaces. It is a logical line consisting of spaces, tabs, formfeeds or comments that are basically ignored.

Using blank lines in top-level-functions and classes:

```
class my_first_class:
    pass
class my_second_class:
    pass
def top_level_function():
    return None
```

Adding two blank lines between the top-level-functions and classes will have a clear separation and will add more sensibility to the code.

Using blank lines in defining methods inside classes:

```
class my_class:
    def method_1(self):
        return None

    def method_2(self):
        return None
```

Here, a single vertical space is enough for a readable code.

You can also use blank spaces inside multi-step functions. It helps the reader to gather the logic of your function and understand it efficiently. A single blank line will work in such case.

An example to illustrate such:

```
def calculate_average(number_list):
    sum_list = 0
    for number in number_list:
        sum_list = sum_list + number

    average = 0
    average = sum_list / len(number_list)


   return average
```

Above is a function to calculate the **average**. There is a blank line between each step and also before the **return** statement.

The use of blank lines can greatly improve the readability of your code and it also allows the reader to understand the separation of the sections of code and the relation between them.

Naming Conventions

Choosing names which are sensible and can be easily understandable, while coding in Python, is very crucial. This will save time and energy of a reader. Inappropriate names might lead to difficulties when debugging.

Naming Styles

Naming variables, functions, classes or methods must be done very carefully. Here's a list of the type, naming conventions and examples on how to use them:

Type	**Naming Conventions**	**Examples**
Variable	Using short names with CapWords.	T, AnyString, My_First_Variable
Function	Using a lowercase word or words with underscores to improve readability.	function, my_first_function

Type	**Naming Conventions**	**Examples**
Class	Using CapWords and do not use underscores between words.	Student, MyFirstClass
Method	Using lowercase words separated by underscores.	Student_method, method
Constants	Using all capital letters with underscores separating words	TOTAL, MY_CONSTANT, MAX_FLOW
Exceptions	Using CapWords without underscores.	IndexError, NameError
Module	Using short lower-case letters using underscores.	module.py, my_first_module.py
Package	Using short lowercase words and underscores are discouraged.	package, my_first_package

Choosing names

To have readability in your code, choose names which are descriptive and give a clearer sense of what the object represents. A more real-life approach to naming is necessary for a reader to understand the code.

Consider a situation where you want to store the name of a person as a string:

```
>>> name = 'John William'
>>> first_name, last_name = name.split()
>>> print(first_name, last_name, sep='/ ')
John/ William
```

Here, you can see, we have chosen variable names like **first_name** and **last_name** which are clearer to understand and can be easily remembered. We could have used short names like **x**, **y** or **z** but it is not recommended by PEP 8 since it is difficult to keep track of such short names.

Consider another situation where you want to double a single argument. We can choose an abbreviation like **db** for the function name:

```
# Don't
def db(x):
    return x * 2
```

However, abbreviations might be difficult in situations where you want to return back to the same code after a couple of days and still be able to read and understand. In such cases, it’s better to use a concise name like **double_a_variable**:

```
# Do
def double_a_value(x):
    return x * 2
```

Ultimately, what matters is the readability of your code.

Comments

A comment is a piece of code written in simple English which improves the readability of code without changing the outcome of a program. You can understand the aim of the code much faster just by reading the comments instead of the actual code. It is important in analyzing codes, debugging or making a change in logic.

Block Comments

Block comments are used while importing data from files or changing a database entry where multiples lines of code are written to focus on a single action. They help in interpreting the aim and functionality of a given block of code.

They start with a **hash(#)** and a single space and always indent to the same level as the code:

```
for i in range(0, 10):

    # Loop iterates 10 times and then prints i

    # Newline character

    print(i, '\n')
```

You can also use multiple paragraphs in a block comment while working on a more technical program.

Block comments are the most suitable type of comments and you can use it anywhere you like.

Inline Comments

Inline comments are the comments which are placed on the same line as the statement. They are helpful in explaining why a certain line of code is essential.

Example of inline comments:

```
x = 10  # An inline comment

y = 'JK Rowling' # Author Name
```

Inline comments are more specific in nature and can easily be used which might lead to clutter. So, PEP 8 basically recommends using block comments for general-purpose coding.

Document Strings

Document strings or docstrings start at the first line of any function, class, file, module or method. These type of comments are enclosed between single quotations (''') or double quotations (""").

An example of docstring:

```
def quadratic_formula(x, y, z, t):
    """Using the quadratic formula"""
    t_1 = (- b+(b**2-4*a*c)**(1/2)) / (2*a)
    t_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)


    return t_1, t_2
```

Whitespaces in Expressions and Statements

In computing, whitespace is any character or sequence of characters which are used for spacing and have an ‘empty’ representation. It is helpful in improving the readability of expressions and statements if used properly.

Whitespace around Binary Operators

When you’re using assignment operators (**=, +=, -=,and so forth**) or comparisons (**==, !=, >, <. >=, <=**) or booleans (**and, not, or**), it is suggested to use a single whitespace on the either side.

Example of adding whitespace when there is more than one operator in a statement:

```
# Don't

b = a ** 2 + 10

c = (a + b) * (a - b)
```

```
# Do
b = a**2 + 10
c = (a+b) * (a-b)
```

In such mathematical computations, you should add whitespace around the operators with the least priority since adding spaces around each operator might be confusing for a reader.

Example of adding whitespaces in an **if** statement with many conditions:

```
# Don't
if a < 10 and a % 5 == 0:
    print('a is smaller than 10 and is divisible by 5!')


# Do
if a<10 and a%5==0:
    print('a is smaller than 10 and is divisible by 5!')
```

Here, the **and** operator has the least priority, so whitespaces have been added around it.

Colons act as binary operators in slices:

```
ham[3:4]

ham[x+1 : x+2]

ham[3:4:5]

ham[x+1 : x+2 : x+3]

ham[x+1 : x+2 :]
```

Since colons act as a binary operator, whitespaces are added on either side of the operator with the lowest priority. Colons must have the same amount of spacing in case of an extended slice. An exception is when the slice parameter is omitted, space is also omitted.

Avoiding Whitespaces

Trailing whitespaces are whitespaces placed at the end of a line. These are the most important to avoid.

You should avoid whitespaces in the following cases—

Inside a parentheses, brackets, or braces:

```
# Do
list = [1, 2, 3]


# Don't
list = [ 1, 2, 3, ]
```

Before a comma, a semicolon, or a colon:

```
x = 2
y = 3


# Do
print(x, y)


# Don't
print(x , y)
```

Before open parenthesis that initiates the argument list of a function call:

```
def multiply_by_2(a):
    return a * 2
```

```
# Do
multiply_by_2(3)


# Don't
multiply_by_2 (3)
```

Before an open bracket that begins an index or a slice:

```
# Do
ham[5]


# Don't
ham [5]
```

Between a trailing comma and a closing parenthesis:

```
# Do
spam = (1,)


# Don't
spam = (1, )
```

To adjust assignment operators:

```
# Do
variable_1 = 5
variable_2 = 6
```

```
my_long_var = 7


# Don't

variable_1    = 5

variable_2    = 6

my_long_var  = 7
```

Programming Recommendations

PEP 8 guidelines suggest different ways to maintain consistency among multiple implementations of Python like **PyPy**, **Jython** or **Cython.**

An example of comparing **boolean** values:

```
# Don't

bool_value = 5 > 4

if bool_value == True:

return '4 is smaller than 5'


# Do

if bool_value:

return '4 is smaller than 5'
```

Since **bool** can only accept values **True** or **False**, it is useless to use the equivalence operator **==** in these type of **if** executions. PEP 8 recommends the second example which will require lesser and simpler coding.

An example to check whether a list is empty or not:

```
# Don't
```

```
list_value = []
if not len(list_value):
    print('LIST IS EMPTY')


# Do
list_value = []
if not list_value:
    print('LIST IS EMPTY')
```

Any **empty list** or string in Python is falsy. So you can write a code to check an empty string without checking the length of the list. The second example is more simple, so PEP encourages to write an **if** statement in this way.

The expression **is not** and **not ... is** are identical in functionality. But the former is more preferable due to its nature of readability:

```
# Do
if x is not None:
    return 'x has a value'


# Don't
if not x is None:
    return 'x has a value'
```

String slicing is a type of indexing syntax that extracts substrings from a string. Whenever you want to check if a string is prefixed or suffixed, PEP recommends using **.startswith()** and **.endswith()** instead of list slicing. This is because they are cleaner and have lesser chances of error:

```
# Do
if foo.startswith('cat'):


# Don't
if foo[:3] == 'cat':
```

An example using **.endswith()**:

```
# Don't
if file_jpg[-3:] == 'jpg':
    print('It is a JPEG image file')


# Do
if file_jpg.endswith('jpg'):
    print('It is a JPEG image file')
```

Though there exists multiple ways to execute a particular action, the main agenda of the guidelines laid by PEP 8 is simplicity and readability.

When to Ignore PEP 8

You should never ignore PEP 8. If the guidelines related to PEP8 are followed, you can be confident of writing readable and professional codes. This will also make the lives of your colleagues and other members working on the same project much easier.

There are some exclusive instances when you may ignore a particular guideline:

- After following the guidelines, the code becomes less readable, even for a programmer who is comfortable with reading codes that follow PEP 8.
- If the surrounding code is inconsistent with PEP.

- Compatible of code with older version of Python is the priority.

Checking PEP 8 Compliant Code

You can check whether your code actually complies with the rules and regulations of PEP 8 or not. **Linters** and **Autoformatters** are two classes of tools used to implement and check PEP 8 compliance.

Linters

It is a program that analyzes your code and detects program errors, syntax errors, bugs and structural problems. They also provide suggestions to correct the errors.

Some of the best linters used for Python code:

- pycodestyle is a tool to verify the PEP 8 style conventions in your Python code.

You can run the following from the command line to install **pycodestyle** using **pip**:

```
pip install pycodestyle
```

To display the errors of a program, run **pycodestyle** in this manner:

```
pycodestyle my_code.py

my_code.py:1:11: E231 missing whitespace after '{'

my_code.py:3:19: E231 missing whitespace after ')'

my_code.py:4:31: E302 expected 2 blank lines, found 1
```

- flake8 is a Python wrapper that verifies **PEP 8**, **pyflakes**, and circular complexity.

Type the command to install **flake8** using **pip**:

```
pip install flake8
```

Run **flake8** from the terminal using the command:

```
flake8 calc.py

calc.py:24:3: E111 indentation is not a multiple of two

calc.py:25:3: E111 indentation is not a multiple of two

calc.py:45:9: E225 missing whitespace around operator
```

You can also use some other good linters like pylint, pyflakes, pychecker and mypy.

Autoformatters

An autoformatter is a tool which will format your code to adapt with PEP 8 automatically.One of the most commonly used autoformatter is black.

To install **black** using **pip**, type:

```
pip install black
```

Remember, you need to have Python 3.6 or above to install **black**.

An example of code that doesn't follow PEP 8:

```
def add(a, b): return a+b


def multiply(a, b):
    return \
      a   * b
```

Now run black following the filename from the terminal:

```
black my_code.py

reformatted my_code.py

All done!
```

The reformatted code will look like:

```
def add(a, b):
    return a + b


def multiply(a, b):
    return a * b
```

Some other autoformatters include autopep8 and yapf. Their work is similar to **black**.