python assignment

February 26, 2023

[ ]:
```
# Q1  Who developed Python Programming Language?
```

[ ]:
```
'''
Python was created by Guido van Rossum in the late 1980s while he was working␣
 ↪at the National Research Institute for
Mathematics and Computer Science in the Netherlands. Van Rossum started working␣
 ↪on Python in December 1989,
and its first public release, Python 0.9.0, came out in
February 1991. Since then, Python has been continuously developed by a␣
 ↪community of volunteers and maintained
by the Python Software Foundation.
'''
```

[ ]:
```
# Q2 Whch type OF Programming does Python support?
```

[ ]:
```
'''
Python supports several programming paradigms, including:
1.Procedural programming
2.Object-oriented programming (OOP)
3.Functional programming
4.Aspect-oriented programming (AOP)
Python's support for multiple paradigms makes it a versatile programming␣
 ↪language that can be used for a wide range
of applications, from web development to scientific computing and data analysis.
'''
```

[ ]:
```
# Q3 Is Python case- sensitive when dealing with identifiers?
```

[ ]:
```
'''
Yes, Python is case-sensitive when dealing with identifiers such as variable␣
 ↪names, function names, and class names.
This means that identifiers with different capitalization are considered␣
 ↪different and distinct.

For example, the variable name "age" is not the same as "Age" or "AGE".
            They are all considered different variables in Python.
```

```
It's important to be consistent with capitalization when using identifiers in␣
 ↪Python to avoid errors and confusion in your code.
'''
```

[ ]:
```
# Q4 What is the correct extension of  the Python file?
```

[ ]:
```
'''
The correct extension for a Python file is ".py".

When you save a Python program, you should give it a name that reflects its␣
 ↪purpose and
end the file name with the ".py" extension.

For example, if you are creating a Python program to calculate the area of a␣
 ↪circle,
you might name your file "circle_area.py".

This extension tells the operating system and text editor that the file␣
 ↪contains Python code and
should be executed using the Python interpreter.
'''
```

[ ]:
```
# Q5 Is Python code compiled or interpreted?
```

[ ]:
```
'''
Python is an interpreted language, meaning that the source code is executed␣
 ↪directly by an interpreter
without being compiled into machine code beforehand.

When a Python program is executed, the interpreter reads and translates the␣
 ↪code into bytecodes,
which are then executed by the Python virtual machine. This process happens␣
 ↪dynamically at runtime,
allowing for more flexibility and ease of development.

While Python is generally considered to be an interpreted language, there are␣
 ↪tools like PyInstaller,
PyOxidizer, and cx_Freeze that can be used to package Python code into␣
 ↪standalone executables
that can be run on other computers without requiring a Python interpreter to be␣
 ↪installed.
These tools essentially compile the Python code into executable binary files,
but they still rely on the Python interpreter to execute the code.

'''
```

[ ]:
```
# Q6 Name few Blocks of code used to define In Python language
```

[ ]:
```
'''
In Python, there are several block structures used to define code, including:

If statements: used to execute a block of code if a condition is true.
if condition:
    # code to execute if condition is true

For loops: used to iterate over a sequence of values and execute a block of␣
 ↪code for each iteration.
for variable in sequence:
    # code to execute for each iteration

While loops: used to repeatedly execute a block of code as long as a condition␣
 ↪is true.
while condition:
    # code to execute while condition is true

Functions: used to define reusable blocks of code that can be called from other␣
 ↪parts of the program.
def function_name(arguments):
    # code to execute when the function is called
    return value

Classes: used to define user-defined data types and their associated methods.
class ClassName:
    def __init__(self, arguments):
        # code to initialize the object
    def method_name(self, arguments):
        # code to execute when the method is called

These are just a few examples of the block structures used in Python.
There are many more, and they can be combined and nested in various ways to␣
 ↪create complex programs.
'''
```

[ ]:
```
# Q7 State a character used to give single-line comments in Python?
```

[ ]:
```
'''
In Python, a hash character (#) is used to indicate a single-line comment.
Any text that appears after a hash character on a line is ignored by the Python␣
 ↪interpreter and is treated as a comment.
'''
```

[ ]:
```
# Q8 Mention functions which can help us to find the version of python that we␣
 ↪are currently working on?
```

[ ]:
```
'''
There are several functions that can be used to determine the version of Python␣
 ↪that is currently installed and running.
Here are a few examples:

sys.version: This function returns a string containing the version number of␣
 ↪the Python interpreter.
import sys
print(sys.version)

platform.python_version(): This function returns a string containing the␣
 ↪version number of the Python interpreter
and additional information about the platform.
import platform
print(platform.python_version())

sys.version_info: This function returns a tuple containing the major, minor,␣
 ↪and micro version numbers of the Python interpreter.
import sys
print(sys.version_info)

These functions can be useful for determining which version of Python is␣
 ↪installed and for checking compatibility
with libraries and other software that may require a specific version of Python.
'''
```

[ ]:
```
# Q9 Python supports the creation of anonymous functions at runtime, using a␣
 ↪construst called _________________
```

[ ]:
```
'''
Python supports the creation of anonymous functions at runtime, using a␣
 ↪construct called
"lambda functions" or simply"lambda expressions".

Lambda functions are small, single-expression functions that can be defined␣
 ↪inline without a specific name.
They are often used to create short, simple functions that are used only once␣
 ↪and don't require a separate definition.

Lambda functions are created using the lambda keyword, followed by the function␣
 ↪arguments and
a single expression that is evaluated and returned when the function is called.␣
 ↪Here's an example:
# Define a lambda function that adds two numbers
add = lambda x, y: x + y
# Call the lambda function
```

```
result = add(2, 3)
print(result)

# Output: 5

In this example, we define a lambda function called add that takes two␣
 ↪arguments and returns their sum.
We then call the add function with arguments 2 and 3, which returns 5.
'''
```

[ ]:
```
# Q10 What does pip stand for python?
```

[ ]:
```
'''
pip stands for "Pip Installs Packages".
pip is the package installer for Python that allows you to easily install,␣
 ↪upgrade, and
remove third-party packages and libraries from the Python Package Index (PyPI)␣
 ↪or other package repositories.
'''
```

[ ]:
```
# Q11 Mention a few build-in functions in python?
```

[ ]:
```
'''
Python comes with a large number of built-in functions that can be used for a␣
 ↪variety of tasks. Here are a few examples:

print(): used to display text or variables to the console.

len(): used to determine the length of a string, list, tuple, dictionary, or␣
 ↪other iterable object.

input(): used to get user input from the console.

int(), float(), str(): used to convert values between different data types.

range(): used to generate a sequence of numbers.

sorted(): used to sort a list or other iterable object.

type(): used to determine the data type of a value or variable.

abs(): used to get the absolute value of a number.

max(), min(): used to find the maximum or minimum value in a list or other␣
 ↪iterable object.
```

```
sum(): used to add up the elements in a list or other iterable object.

These are just a few examples of the built-in functions available in Python.
There are many more, and you can also create your own functions to perform␣
 ↪specific tasks.
'''
```

[ ]:
```
# Q12 When is the maximum possible length of on identifier in Python?
```

[ ]:
```
'''
In Python, the maximum possible length of an identifier (variable name,␣
 ↪function name, etc.)
is not specifically defined or limited.
However, in practice, there are some practical limitations to the length of␣
 ↪identifiers that you can use.

According to the Python documentation, identifiers can be of any length and can␣
 ↪consist of letters
(both lowercase and uppercase), digits, and underscores. However, they must␣
 ↪begin with a letter or underscore
and cannot contain spaces or other special characters.

In general, it is a good practice to keep your variable and function names␣
 ↪concise
and descriptive, rather than overly long or complicated. This makes your code␣
 ↪easier to read and understand,
and can help avoid errors and typos.

While there is no strict limit on the length of identifiers in Python, it is␣
 ↪generally a good idea to
keep them reasonably short and meaningful. Most code editors and IDEs also␣
 ↪provide autocomplete features
that can help you avoid having to type long names repeatedly.
'''
```

[ ]:
```
# Q13 When are the benefits of using Python?
```

[ ]:
```
'''
There are many benefits to using Python as a programming language. Here are␣
 ↪some of the key advantages:

Easy to learn and use: Python has a simple syntax and a large standard library,␣
 ↪which makes it easy to learn and use.
                        It's a great language for beginners, but also powerful␣
 ↪enough for experienced developers.
```

```
Versatile: Python can be used for a wide range of applications, including web␣
 ↪development, data analysis,
          machine learning, scientific computing, and more.

Open source: Python is an open-source language, which means that it is freely␣
 ↪available to use and distribute.
            It also has a large and active community of developers who␣
 ↪contribute to the language and its ecosystem
            of libraries and tools.

Large ecosystem of libraries and tools: Python has a vast ecosystem of␣
 ↪libraries and tools that can help you get your work
                                        done more efficiently. This includes␣
 ↪libraries for data analysis, machine learning,
                                        web development, scientific computing,␣
 ↪and more.

Cross-platform: Python is a cross-platform language, which means that you can␣
 ↪run the same code on different operating systems,
               such as Windows, macOS, and Linux.

Easy to read and maintain: Python code is easy to read and maintain, which␣
 ↪makes it ideal for collaborative development
                          and long-term projects.

High-level language: Python is a high-level language, which means that it␣
 ↪abstracts away many of the low-level
                     details of computer hardware and memory management. This␣
 ↪makes it easier to write code that
                     is concise and expressive.

These are just a few of the many benefits of using Python as a programming␣
 ↪language.
'''
```

[ ]:
```
# Q14 How is memory managed in Python?
```

[ ]:
```
'''
Memory management in Python is handled automatically by the Python interpreter␣
 ↪using a system called reference counting.
Every object in Python has a reference count, which keeps track of how many␣
 ↪references (i.e., variables, data structures, etc.)
point to the object. When the reference count of an object reaches zero, it is␣
 ↪automatically deleted from memory.
```

```
In addition to reference counting, Python also uses a system called garbage␣
 ↪collection to deal with circular references.
A circular reference occurs when two or more objects refer to each other in a␣
 ↪way that creates a loop.
In this case, reference counting alone would not be sufficient to delete the␣
 ↪objects from memory, as their reference
counts would never reach zero. The garbage collector periodically checks for␣
 ↪circular references and deletes the objects
that are involved in them.

Python also provides a few tools for managing memory manually, such as the gc␣
 ↪module, which allows you to control the
garbage collector, and the sys module, which provides information about the␣
 ↪memory usage of your program.

Overall, Python's memory management system is designed to be efficient and easy␣
 ↪to use, allowing developers to
focus on writing code without worrying too much about memory management.
'''
```

[ ]:
```
# Q15 How to install Python on Windows and set path variables?
```

[ ]:
```
'''
To install Python on Windows and set path variables, follow these steps:

Download the latest version of Python from the official website (https://www.
 ↪python.org/downloads/).
Choose the appropriate version based on your operating system (32-bit or␣
 ↪64-bit) and the version of Python you want to install.

Run the downloaded installer and follow the on-screen instructions to complete␣
 ↪the installation process.
Choose the "Add Python to PATH" option during the installation process to add␣
 ↪Python to your system's PATH environment variable.

To check that Python is installed and the PATH variable is set correctly, open␣
 ↪a command prompt and type "python"
(without the quotes). This should start the Python interpreter, and you should␣
 ↪see the Python version number displayed.

If you encounter any issues with the PATH variable, you can manually add it by␣
 ↪following these steps:

a. Right-click on "My Computer" or "This PC" and select "Properties".
```

```
b. Click on "Advanced system settings" and then click on the "Environment␣
 ↪Variables" button.

c. Under "System Variables", scroll down and select "Path" and click on "Edit".

d. Click on "New" and add the path to the Python installation directory (e.g.,␣
 ↪C:\Python39).

e. Click "OK" to save the changes and close all windows.

f. Open a new command prompt and type "python" to confirm that Python is now in␣
 ↪your system's PATH environment variable.

That's it! You should now have Python installed on your Windows system and the␣
 ↪PATH variable set correctly.
You can start writing and running Python code from the command prompt or using␣
 ↪an integrated development environment (IDE)
like PyCharm, Visual Studio Code, or IDLE.

'''
```

[ ]:
```
# Q16 Is indentation required In python?
```

[ ]:
```
'''
Yes, indentation is required in Python.
In Python, indentation is used to define the scope and hierarchy of code␣
 ↪blocks, such as loops, functions,
and conditional statements. Unlike many other programming languages, which use␣
 ↪braces or other symbols to delimit
code blocks, Python uses indentation to visually indicate which lines of code␣
 ↪are part of a particular block.

The standard convention in Python is to use four spaces for each level of␣
 ↪indentation.
When you write Python code, you need to be careful to use consistent and␣
 ↪correct indentation to avoid
syntax errors or unexpected behavior.

Here's an example of how indentation is used in Python to define a code block:
if x > 0:
    print("x is positive")
    y = x * 2
    print("the value of y is", y)
else:
    print("x is not positive")
    y = x / 2
```

```
    print("the value of y is", y)

In this example, the lines of code that are indented under the if statement are␣
 ↪part of the "if" block,
while the lines of code that are indented under the else statement are part of␣
 ↪the "else" block.
The indentation helps to make the structure of the code clear and easy to read.
'''
```