LEARN PYTHON PROGRAMMING QUICKLY

A BEGINNER'S GUIDE TO CODING WITH PYTHON PROGRAMMING BY LEARNING JUST 5 SIMPLE CONCEPTS. START CODING WITHIN 3 HOURS

CHRIS FORD

LEARN

PYTHO

PROGRAMM
QUICKLY

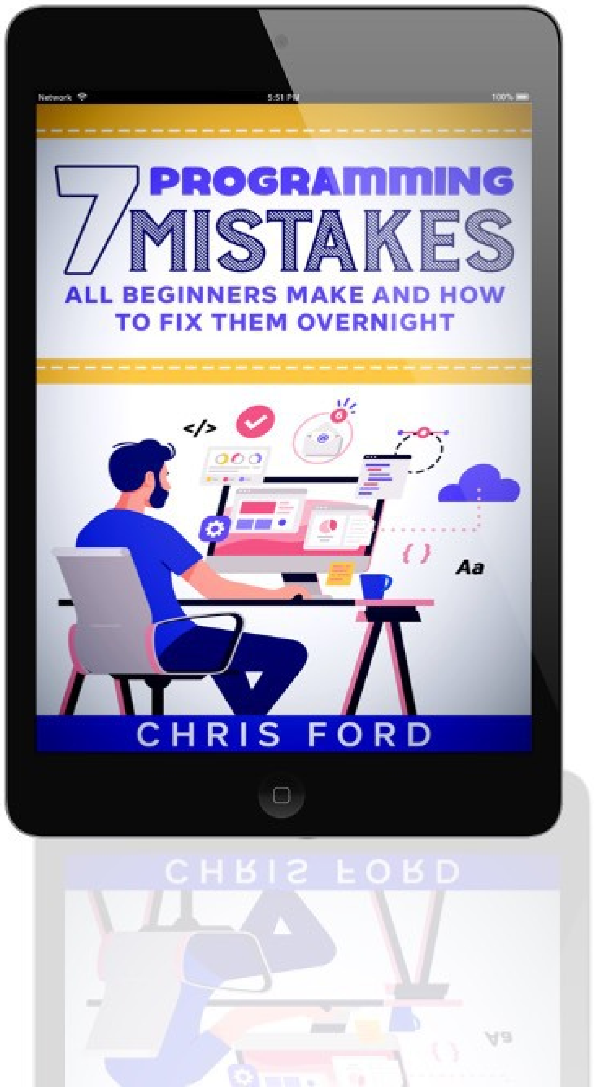
LEARN PYTHON PROGRAMMING QUICKLY

A BEGINNER'S GUIDE TO CODING WITH PYTHON PROGRAMMING BY LEARNING JUST 5 SIMPLE CONCEPTS. START CODING WITHIN 3 HOURS.

CHRIS FORD

CONTENTS

A FREE GIFT TO OUR READERS



Programming can be hard if you don’t avoid these 7 biggest mistakes!
Visit chrisfordbooks.com to get this free PDF guide.

http://www.chrisfordbooks.com/

INTRODUCTION

Computers have changed the world with their arithmetic and logical skills. Computers can also be called dumb machines that can do only what they are said to do. Humans who can communicate efficiently with these dumb machines to do tasks efficiently and quickly are known as computer programmers. Like humans who communicate with each other using different languages, these computer programmers try to communicate with computers using high-level programming languages that they can understand.

Python, Java, C, C#, and Swift are some of these high-level programming languages that have changed the world over the last few decades. All websites, databases, mobile apps, ATMs, traffic control, and many other day-to-day usages use software written using these programming languages.

This book acts as a beginner's guide to programming by using Python as a reference programming language. Python has proved its worth in the programming arena with its high adaptability to any domain or operating system. If you are a beginner and are looking forward to understanding programming basics using Python, you have made the right choice. Even if you are an experienced programmer, alienating from other languages and choosing

Python as your primary programming language can help you to write more effective code in less time.

WHY IS LEARNING PYTHON ESSENTIAL?

Python is considered a programming language that can automate things in any domain related to software technology. Do you want to create a web application? Python has web libraries to make it possible. Do you want to analyze large chunks of data and create a machine learning model to predict future instances? Python can do this. Do you want to create automated scripts that can maintain your web servers and databases? Python can do it gracefully. Do you want to develop exploits and find vulnerabilities that can exploit the system? Python can help you.

This fantastic correlation with every domain that exists out there has made Python a popular programming language to learn. Many software companies have listed Python as their programming language and will hire an employee based on their efficiency with it. Learning Python can help you secure a career in the software industry irrespective of your chosen domain, making it an essential prerequisite among beginners.

WHO AM I?

My name is Chris Ford, and I am an experienced Python programmer specializing in web design and data mining. My love for programming started seven years back when I first discovered a mobile app that automates things. I used it to perform tasks that I often forgot to do. Intrigued by the concept, I started to explore more about programming and the wonders it possesses. Even after strenuous research, because of a lack of good resources I struggled in my initial days to create any productive logical code that could turn into actual software.

I performed due diligence and understood that I have been working with programming languages that are complex and work only for a single purpose. In my quest to find my programming language, I explored various programming languages such as C#, C, C++, and Java, before finally landing on Python. At first, I felt Python was a more straightforward language than the other higher-level languages that I had tried before. Once I got the hang of it, I understood the power it had and how much more it could do with little code.

I was intrigued by the prowess of Python and started to work on small projects that helped me understand the language from a better perspective. Once I got good experience with small projects, I began to work on big projects such as websites and desktop software. My love for Python has grown exponentially over the years and helped me move forward in my programming career. I started to become proficient with web design and data mining with the help of Python libraries over the next couple of years.

Last year due to the nationwide lockdown, when I was stuck like everyone in their own homes, I decided to write a book that could help people who are just being introduced to Python get an introduction that would help them shape their careers. Over the next few months, I did extensive research to create content that could be easily understood by a layman to the subject. This book provides concepts and examples specific to Python. It will help you ignite a programming philosophy that is essential for becoming an efficient programmer.

Teaching Python is not an easy task as it requires a lot of precise understanding of the subject. So, to make readers understand the concepts thoroughly, I have provided practical exercises and many examples that one can use to work independently.

HOW CAN THIS BOOK HELP YOU?

Python programming is simple to understand but hard to implement. To be aware of the abilities of the implementation features of Python, you need to have strong knowledge of the basics of the programming language. This book provides theoretical and practical knowledge required for a programmer to become efficient in writing Python programs.

To get the most out of this book, following the strategy below is recommended.

1. Try to understand the fundamental concepts and use cognitive techniques such as mind maps to recall the content actively.
2. Once you know the basics, start to look at the examples and understand how Python uses its syntax to implement the basic concept into a programmatic sense.
3. Once you understand the examples, solve exercises on your own on your computer. If you encounter errors, then Google them and find out the reasons behind it.
4. Constantly and consistently review the content for becoming an expert in the subject.

Python as a programming language is versatile and requires a lot of attention and persistence to master. I am ready to provide you with information about fundamentals that a programmer needs to be aware of. To get the most out of this book, be patient and stay hopeful about your programming journey.

1

INTRODUCTION TO PYTHON

Python is a high-level, general-purpose programming language. Guido van Rossum created it during Christmas of 1989 out of boredom. He had the core goal of making it appeal to Unix/C hackers. Seeing the project gaining attention in the programming community, he spent much time improving its core implementation. With experience in working as a core developer on CWI (Centrum Wiskunde & Informatica) to create ABC programming language, Rossum adopted ABC's different features to create an interpreted programming language that was easy and intuitive. He named the programming language after Monty Python, a famous British comedy group.

Since its release in 1990, Guido van Rossum has been the Benevolent Dictator for Life in the Python programming community, making him the chairman of project development in open-source terminology. Python has become one of the most popular programming languages with the release of Python 2.0 in 2000. The introduction of features such as garbage collector, list comprehensions, and support for Unicode has made many programmers migrate to Python from more traditional programming languages such as Pearl. Further significant changes in the

Python community came when Python 3.0 was released in 2008.

Python has consistently been mentioned as one of the most innovative and popular programming languages in the programming community. According to TIOBE rankings, it has been in the Top 10 programming languages almost all years from its inception.

Initially, the popularity of Python as a programming language increased mainly due to its motto, “There should probably only be one way to do it.” This contradicted the motto of the then-popular language, Pearl, ''There are many ways to do it.” Python uses a minimalist philosophy and encourages programmers to create code that is not complicated.

Its minimalist approach also helped it get popular among beginners; it's taught as a reference programming language for first-year undergraduate students in universities worldwide. Before the invention of Python, programmers mainly used the high-level programming languages for scientific development. With the entrance of Python, real-world software development has increased exponentially.

The Python core team has expanded its ability to create software applications in different domains. Some of these applications are provided below to help readers understand the magnanimous impact of Python on software engineering.

1. Python is extensively used as a scripting language for developing web applications. Python third-party frameworks such as Django and Tornado can create complex applications that can provide both reliability and security to the website's visitors. Python is highly

implemented in various popular websites such as Facebook, Netflix, and Google for multiple implementations. For example, Twisted, a framework used by Dropbox engineers, can help companies effectively interact with their computers in their network.

2. Python is also extensively used in scientific computing areas. It has libraries such as Numpy and Scipy that programmers can use to create applications depending on different mathematical areas such as geometry and calculus. Python is also a significant resource for programmers working with digital vision and image processing applications.
3. Python has also made its mark in the now evolving artificial intelligence and machine learning stream. Libraries such as Tensorflow and Sci-kit learn are reliable resources for developing and experimenting with machine learning models. Python's adaptability to natural language processing also helps companies use it to create automatic translators.
4. The impact of Python on software development is enormous. It helped developers create complex software such as GIMP, Blender, and Maya. It also has decent adaptability for making mobile video games.
5. Python is the preferred scripting language for Devop engineers who usually have to deal with a ton of Linux software. Developers write almost all Linux installation tools in Python for better portability among different distros. macOS, the popular operating system provided by Apple, also uses Python to write various default software.
6. The adaptability of Python to both black and white hackers is also commendable. Many black hat hackers use Python to create exploits that can automatically find vulnerabilities in a flawed system.

Penetration testers also use Python for exploit development that can detect these flaws way before the hackers do.

7. Python is also a preferred programming language for creating automated tools such as bots and scrapers to reduce manual work for programmers.

With its huge impact on software engineering, Python has influenced modern programming languages in different aspects. Developers of Go, Groovy, and Swift, inspired by Python, implemented minimalistic philosophy in their way while deploying these languages.

THE BENEFITS OF LEARNING PYTHON

During the great internet boom of the 1990s, many software companies employed programmers to write complex code to meet demand from individuals worldwide. However, the development cycle of software increased tremendously as the code started to become complicated. Most of the development time had gone into code maintenance and resolving bugs. During this time, Python was a lifesaver for thousands of programmers, enabling them to create a simplified coding style to help them maintain and deploy new code quickly.

Understanding the different benefits that Python provides for its users is essential for beginners to take note of.

1. Interpreted programming language

Python uses an interpreter to execute various instructions that a programmer provides directly. It parses the code during this process and creates an intermediary machine-level language to reduce the waiting time during execution. Due to being a high-level language, Python uses natural language elements that make the code easier to use and automate.

2. Open source

Python is an open-source language, and hence anyone can modify and distribute it without any problem. Its open-source nature is the main reason for its success in the real world. Python has always been in the top three programming languages in GitHub. Open-source culture also helps programmers save time by not creating code that has already been written.

Beginners can also use these open-source projects to understand Python's complex programming logic behind software development.

3. Dynamically typed

Unlike static programming languages that use compilation time to run their code, dynamically typed languages use a runtime mechanism. While both static and dynamic typing languages have their advantages and disadvantages, dynamically typed programming languages such as Python are believed to be more reliable and reusable, according to experienced computer scientists.

4. It supports multiple paradigms

Any high-level programming language needs to support programming paradigms to reuse the already written code effectively. Python provides all three popular paradigms for its users: structured, functional, and object-oriented. Even though Python provides multiple paradigm opportunities for its users, it is always recommended to use a consistent paradigm for better code maintainability.

5. It uses garbage collection mechanism

Every programming language uses its strategy for effective memory management. Python uses garbage collectors to use memory that is no longer referenced. Garbage collectors can help programmers interact with data that has just been deleted or modified to increase the program's speed.

6. Excellent software quality

Python's success is mainly due to its readability. All the code written supports coherence and is hence easily maintainable. Written Python code is also always uniform and therefore the complex logic involved in the software development is easy to understand. Software quality also increases developer productivity by taking less time to debug the code. For example, it only takes a third of the code to rewrite a C language parsing library using Python.

7. Portability

Python is developed in a way that it can run in any operating system or platform with a simple installation of its interpreter. Its extreme portability has helped programmers create portable GUIs (Graphical User Interfaces) for various standalone applications.

If you have written a program to run in a Linux system using Python, you can quickly change it by adding a few lines to work in either Windows or macOS.

8. Huge standard and custom libraries

There is no denying that the popularity of Python is most probably due to its extensive standard and custom libraries that programmers can use to create any application in any domain.

All the standard libraries can help programmers create and interact with the system at an application level, whereas portable libraries can help programmers develop applications in different domains such as data analysis and machine learning.

9. Component integration

Python can integrate itself into any pre-written code or application, and hence it is a preferred programming language for customization. Its extension capabilities are intense and can help programmers interact with any traditional programming language and library.

Its extension capabilities have helped developers create new custom interfaces for various legacy software.

10. Supportive community

Python has a very supportive community that helps beginners not get overwhelmed by the enormous number of libraries available. There are many forums where programmers can ask questions when they are stuck with an error while deploying or debugging the code.

Resources related to standard and custom libraries are also excellent for beginners to start working on projects. A vast number of open-source projects available in GitHub can also help programmers write their code using the same principles.

```
link: function(scope, element, attr, ngSwitchController) {
  var watchExpr = attr.ngSwitch || attr.on,
      selectedTranscludes = [],
      selectedElements = [],
      previousElements = [],
      selectedScopes = [];

  scope.$watch(watchExpr, function ngSwitchWatchAction(value) {
    var i, ii;
    for (i = 0, ii = previousElements.length; i < ii; ++i) {
      previousElements[i].remove();
    }
    previousElements.length = 0;

    for (i = 0, ii = selectedScopes.length; i < ii; ++i) {
      var selected = selectedElements[i];
      selectedScopes[i].$destroy();
      previousElements[i] = selected;
      $animate.leave(selected, function() {
        previousElements.splice(i, 1);
      });
    }

    selectedElements.length = 0;
    selectedScopes.length = 0;

    if ((selectedTranscludes = ngSwitchController.cases['!' + value] || ngSwitchC
      scope.$eval(attr.change);
      forEach(selectedTranscludes, function(selectedTransclude) {
        var selectedScope = scope.$new();
        selectedScopes.push(selectedScope
```

DIFFERENT VERSIONS OF PYTHON

Python 2 and Python 3 are the two popular versions that beginners can use to create real-world applications. Understanding the differences between them in detail can help you while working with a real-world project or for a client.

Python 2

Python 2 was first released in 2000 with various innovative features that helped it become a preferred choice for programmers who craved simplicity.

In 2021, the Python 2.7 version has become a legacy software, and no further development will be provided to it, making it an impractical choice for programmers. However, there are programmers who are stuck with Python 2.7 due to difficulties migrating written code into Python 3. Many companies and libraries don't migrate their code instantly because it requires both money and time.

Many notable features available in Python 3 have been backported by the Python developer team to Python 2.7 for migrating the projects swiftly. Researchers say that even though Python 2.7 has become a legacy software, there will be a demand for Python 2 developers until 2025.

Python 3

Python 3 was released in 2008 and since then has gone through significant changes to make Python as effective as possible for real-world projects. Unlike Python 2, it uses simpler syntax and hence can be easy to maintain.

The popularity of Python 3 is also due to its adaptability with different modern computing fields such as artificial intelligence and machine learning. All the developers are creating libraries specifically for Python 3, which is why it is essential to learn it. Python 3.9 is the latest available version for developers.

What Should I Choose?

While the Python version you choose is entirely a personal choice according to your requirements, I recommend Python 3. Python 3 is not only simpler to learn but also helps programmers create more reliable code. Python 3 is also more versatile and powerful, and hence it is my recommended choice for anyone beginning with Python.

Note : All the code provided in this book uses Python 3 syntactical rules.

HOW TO INSTALL PYTHON TO GET STARTED

Python is highly portable and can run on any operating system. Head over to Python.org to download Python 3.

Any program or application that you develop can run in all the operating systems with a slight change in the code.

For example, a Python network mapper written for the Windows platform can also work with Linux if you can change the implementation of ports in the software as both operating systems use different approaches to interact with ports.

As a programmer, it is essential that you understand installing Python in different operating systems.

Python on Linux

Linux is an open-source operating system that is majorly used by programmers. As Linux is specifically designed for programming, almost all Linux distros have Python pre-installed in it.

To check whether or not Python is installed in your Linux system, you need to enter a terminal. To open a new terminal in your system, use Ctrl + Alt + N.

Once a terminal is opened, enter the following command.

Terminal Code :

```
$ python3
```

If Python is installed in your system, you will be welcomed with license information related to Python. However, if you receive a blank screen, it means that the Python 3 is not shipped with your Linux distro. You will need to use package managers to install Python.

You need to first upgrade all your programs before trying to install new software. Upgrade all the software in your Ubuntu Linux system using the upgrade command and then proceed with the installation of Python.

Terminal Code :

```
$ sudo apt-get upgrade
$ sudo apt-get install python3
```

Many other popular Linux distros, such as Arch, Gentoo and Kali, use different commands and package managers to install a package. Make sure you research according to your Linux distro not to face any errors.

Python on macOS

macOS is one of the popular operating systems that Apple maintains. Like Linux, macOS also deploys its operating systems with inbuilt Python support.

To check whether or not Python is installed in macOS, you need first to enter a terminal. Open a terminal by entering the Utilities menu in your home bar.

Once a terminal is opened, enter the below command.

Terminal code :

```
$ python3
```

If you are not welcomed by a license message, then Python is not installed in your system. It will help if you use a package manager such as Homebrew to install the latest version of Python in macOS.

Terminal code :

```
$ brew install python3
```

Python on Windows

Windows is the popular operating system in the world that Microsoft developed. Windows systems are not available as default software in Windows, and hence you need to install an executable package to start experimenting with Python programs.

To install Python 3 in Windows, head over to Python.org and click on the downloads tab, where you can find different executable files according to your operating system. Download the executable installer which automatically installs Python according to your preferred location. After installing the Python installer, make sure that you add “path” in environment variables using a control panel to avoid conflicts with any other programming language interpreters present in the system.

You can confirm whether or not a Python interpreter is installed in a Windows system using the below command.

Command prompt code :

```
> python --version
```

If you face any problems while installing Python in your preferred operating system, we suggest you Google the error displayed in the terminal.

2

BASICS OF PYTHON

To create software applications using Python, you need to utilize a development environment to perform high-level programming tasks. This chapter is designed to help readers understand the importance of a productive development style that Python programmers should focus on.

PYTHON INTERPRETER

Python is an interpreted language. It comes with additional advantages that other high-level programming languages cannot provide. For example, Python programs are easy to test as they involve less waiting time.

Compared to its predecessor compilers, interpreters can directly perform actions and are recommended as a favorable choice for beginners.

When you install Python in your system, an interpreter along with an IDLE (Integrated Development and Learning Environment) will be installed in your system. You can either search it in your system or enter the command “python” to trigger IDLE in the shell terminal.

Python IDLE can be considered an ideal development tool for beginners who are just starting with Python. IDLE first reads the line in a program, evaluates its instructions, prints them on the screen, and loops into the following line. This mechanism is known as REPL and is a fundamental concept all interpreters use to parse and run the code effectively.

With the IDLE, you can easily edit and create files with a .py extension. IDLE provides tracebacks to mention the errors in a particular line to keep beginners from getting overwhelmed with failed executions.

How to Use the Python IDLE shell?

Open a terminal using the default “python” command in the terminal.

Terminal code :

```
$ python
```

The terminal will open a shell window, as shown below.

Output :
```
>>>
```

Some examples*:*

IDLE can be used to test Python code with any simple arithmetic calculations and for printing statements.

Program code :
```
>>> print ( “ This is a sample text “ )
```

When you press enter, the IDLE will enter into REPL mode and will print the given text on the screen, as shown below.

Output :
```
This is a sample text
```
In the above example, print() is a method in Python’s standard library that programmers can use to output strings in the terminal window.

Now, we will look at some of the arithmetic examples.

Program code :
```
>>> 2 + 5
```

Here, we are using an addition operator, and the IDLE will instantaneously display seven as output once we press the Enter key.

Output :
```
7
```

Note: Remember that once you start a fresh terminal, all the ongoing code in the IDLE will destroy itself.

How to Open Python Files in IDLE

Python inbuilt text editor provides you three essential features that are essential for a coding environment.

1. It can automatically highlight your code syntax
2. It can automatically help you complete default code
3. It can automatically indent your code

To open a file in IDLE, all you have to do is choose the File menu in the toolbar and click open. Action will open your file system so that you can choose the file. Advanced programmers usually specify a path to open Python files instantly.

How to Edit Files in IDLE

Once you successfully open a Python file, you will be able to edit it with your code. IDLE will provide specific information such as the file name, path, and line and column numbers to help the programmer easily create and debug the code.

After editing the file, you can execute it using the F5 key.

Python IDLE is also a great debugging tool for small projects. It provides essential debugging features such as breakpoints and catching exceptions to debug the code quickly. However, if the volume of the project increases, IDLE will struggle to parse and debug the code effectively.

Note: Python IDLE provides a help() function to provide easy reference for programmers. You can use help() to find implementation information about any method or standard library.

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

While Python IDLE is an excellent way to experiment with small snippets of code, it becomes extremely tough to organize all the written code for Python projects. To solve the problem of messy organization of code and tight integration capabilities, computer programmers use advanced software applications known as IDE (Integrated development environments).

IDE not only helps programmers to create neatly organized code but also provides developer productivity. It consists of built-in development tools such as code editor, debugger, and automation tools to decrease the setup time that usually involves building and migrating code with individual applications.

Some of the features IDEs offer:

1. Easy integration with libraries and frameworks

IDEs make it easy for programmers to integrate with open-source or private frameworks and libraries easily. Some IDEs also provide options to integrate with online repositories such as GitHub directly.

2. Tools for object-oriented programmers

Object-oriented programming is a highly preferred programming paradigm for team projects. IDEs provide class and object browsers to help programmers easily interact with different components in a large project. Several IDEs also provide the option of class hierarchy diagrams as a reference to kick start the project.

3. Syntax highlighting

Syntax highlighting is an important feature that helps programmers to determine different programming structures and reserved keywords quickly. Programmers can also use syntax highlighting to detect errors more swiftly and save time by not encountering build-time errors.

4. Code Completion

IDEs provide intelligent code completion features to help beginners not get deviated from the logical implementation of programs. IDEs use complex artificial intelligence and machine learning algorithms to predict the following procedure in a programming task. However, remember not to rely on them as programming is a human task entirely, and it always will be.

5. Version Control

Libraries and frameworks usually get updated with time, and API changes sometimes can break the application. To help programmers effectively maintain the software applications, IDEs provide version control and automatic refactoring. Using these features, programmers can manage changes to the program without messing with the core functionalities.

Apart from these features, most IDEs provide debugging and customization options to make programming practical and an exciting and enriching experience.

Python is also fortunate to have well-developed IDEs that can facilitate any need by a programmer. Pycharm, Spyder, and eclipse are some popular IDEs that programmers can use to develop Python applications.

PYCHARM

Pycharm is an exclusive Python IDE developed by Jetbrains to create and collaborate with Python projects. Using Pycharm, developers can create concise and readable code, all the while following Python default coding conventions.

Pycharm is also a cross-platform IDE that programmers can install in Windows, macOS, and Linux. It offers support for both Python 2 and Python 3. Pycharm provides both community and professional versions for its users. The community version is free and offers a lot of basic functionalities with minimal customization capabilities. However, if you want to interlink your project with web or scientific frameworks, there is a professional version that charges a fee.

What Are the Exclusive Features of Pycharm?

1. Code editor

Pycharm has an inbuilt code editor that helps programmers create Python code that is of high quality. It provides automatic code suggestions and provides a high-level color scheme to detect different types of syntax easily.

An editor can also be customized using different themes and plugins to improve your productivity and satisfy your visual aesthetics.

The editor is created so that a programmer can easily detect errors and warnings without interpreting them at once. Code regeneration and automatic indentation are also some of the additional features that Pycharm provides.

2. Code Navigation

Pycharm provides a tremendous organizational ability for programmers to structure large projects effectively. Specific features such as bookmarks and lens mode can help beginners structure the code in a way that is intended.

3. Advanced Refactoring

Refactoring is an important feature to be aware of, especially to implement changes related to local and global files. Refactoring can help programmers quickly rename files all at once from classes, objects, variables, and strings.

Pycharm provides advanced refactoring options to customize the code as quickly and effectively as possible without making any build errors.

4. Integration with web technologies

The professional version of Pycharm provides easy integration with web technologies such as HTML, CSS, and

Typescript. Pycharm integrates all these technologies to work with Python frameworks and modules intuitively.

Pycharm also provides live edit features to help Python web developers follow the changes in the inbuilt web browser instantaneously.

5. Integration with scientific libraries

Pycharm has provided tight integration with open-source data science and machine learning libraries such as NumPy and Anaconda in its professional version as Python has high adaptability with data science projects.

6. Software testing

Pycharm also provides integration with unit testing frameworks such as Nose to help programmers involved in big projects. Visual debugger, database integration, and remote development capabilities can maximize the productivity of programmers involved in teams.

WORKING WITH PYCHARM

Step 1: Installing Pycharm

Installation of Pycharm is an easy process, and you can download it with a click on all operating systems. Linux users may need to install Pycharm from their respective package managers.

Head over to Jetbrains official website, click on the download section and wait for the website to detect your operating system. The website will provide an executable file that you can use to install it in your system.

If you are willing to purchase a professional version of Pycharm, you will have to provide your billing details so that your IDE will function as it is expected to.

Once downloaded, install the executable file in your system and wait for it to open automatically. When running on old computers, Pycharm may not detect your Python version automatically, so you may need to enter the path to make everything work as expected.

Step 2: Creating a New Project

When Pycharm is opened, a pop-up will appear showing recent projects. In the top left, you will find an option called "File" on the toolbar which can make a new Python project for you. Click on it, and the Pycharm interface will ask you to select the interpreter you want to use for the project. Select the default option 'virtualenv' for now and head over to the editing interface.

Step 3: Organizing your files in Pycharm

Once a project is opened, you need to create a new folder to organize all your script files. The organization is essential, primarily if you use an object-oriented paradigm to create your software applications.

```
Click New -> Folder to create a new folder
containing all the scripts or classes you
need to.
You can also create a new file using, New ->
File -> Python .
```

The above action will create a file with a .py extension. You can easily rename it according to your requirements.

Step 4: Analyze the interface in Pycharm

1. Pycharm can automatically save your code. However, if you still want to save your files manually, you can

use ctrl + S if you are a Windows user. If you are a Mac user you can use Command + S.
2. You can run and interpret your Python code using Shift+ F10
3. In Windows, using Ctrl + F, you can easily find any small snippet or method in your Python project. If you are a Mac user you can use Command + F.
4. To debug the code, you need to enter “Shift + F9”. Pycharm debugger provides breakpoints and line by line debugging options to easily interact with the code and find the logical problems involved.

PYTHON STYLE GUIDE

Python is a programming language that favors simplicity instead of complexity. To understand Python's essence as a programming language, readers must read Tim Peters’ “The Zen of Python.” It provides 19 guiding principles that dictate the Python style for all projects.

Understanding these principles and following them in their programming workflow can help Python programmers create code that other programmers can easily maintain and understand.

You can find all the 19 working principles related to the Python style guide using the below command in your terminal.

Terminal code :

```
$ import this
```

Here are the critical five styling principles that Python programmers need to be aware of.

1. Beautiful is better than ugly

All Python programmers are suggested to write beautiful semantically code instead of code that is hard to read and understand. Here ugly refers to code that is not clean and unreadable. Remove unnecessary statements or conditionals that can complicate the code. Use indentations for better understanding the code according to the structure. Beautiful code not only rewards you with good readability but also helps to decrease the runtime required.

2. Explicit is better than implicit

Never try to hide the functionality of the code to make it explicit unless required. Being verbose makes understanding the programming logic difficult. This styling principle is also mentioned to encourage Python programmers to write open-source code that can be redistributed.

3. Simple is better than complex

Make sure that you write code that approaches the logic more straightforwardly instead of being complex. If you can solve a problem with ten conditional statements or with one method, solve it with one. Make it as simple as possible.

4. Complex is better than complicated

It is not always possible to write simple code that can do complex actions. To perform complex actions, strive to write complex code instead of complicated code. Always write code in a way that you can catch exceptions whenever something unexpected happens. Exceptions can help you quickly debug your code.

5. There should be only one way to do it

Python programmers are suggested to use only one programming logic for a problem all around the project instead of using three or four logics in different instances. Maintain uniformity to let others easily understand your program code. Uniformity provides flexibility which can lead to easy and quick maintainability.

PYTHON COMMENTS

Comments are introduced to help programmers easily remember or share the programming logic used for a program with other team members while working during projects. Nowadays, most libraries and frameworks work with open-source systems, and it is practical to start using comments so that others can understand your code concisely.

Apart from this reason, comments are also highly recommended to write lines that can be ignored by the interpreter while testing the software application. As a Python programmer, you need to start using comments whenever possible. Comments can also help you while updating the code directories in the future.

There are two types of comments Python programmers can use to make their code clearer and more understandable.

1. Single line comments

Anything followed by # (hash symbol) in a Python program can be called a single line comment.

Program code :

```
# This is a comment, and we will print a
statement
Print (“We have learned about comments”)
```

Output :

```
We have learned about comments
```

Only print statement is executed, and the interpreter has ignored everything that is written after #.

Programmers can use single comments, especially in the middle of the code, to describe program logic or while initiating conditionals and loops.

2. Multi-line comments

You can use hash (#) to create multi-line comments, but it is not considered a good practice. To create multi-line comments, a Python programmer can use string literals.

Program code :

```
‘ ’ ’
This is a comment that can be used to write
sentences or paragraphs.
‘ ’ ’
```

Print (“We have just learned about multi-line comments”)

Output :

```
We have just learned about multi-line
comments
```

Only the print statement is executed, and the interpreter ignores everything that is in between string literals.

Note: Remember not to be redundant while writing comments. Write only what your team members or your future self may need.

3

VARIABLES

To maximize the potential of Python as a programming language, programmers need to start using variables and operators in their programs. Variables and operators are fundamental building blocks of programs that can help build programming logic and create applications that can manipulate memory effectively.

UNDERSTANDING VARIABLES

Software applications usually deal with data. Users upload or download data from applications to completely experience the product. All these use cases have become possible only with the help of variables.

Computer scientists developed variables to effectively store data to a specific location in the computer memory. Variables can be called labels and are usually the most basic unit in a program.

You might have already used variables in mathematics while working with algebra.

Example :

```
2x + 3y
```

```
x = 3 and y = 4
```

Variables in mathematics are usually used to replace a value in an equation. Computer scientists adopted the same concept to effectively manage memory resources in the initial stages of computer development.

To understand what happens when a Python program is executed, read the below example.

Program code :

```
print (" This is an example ")
```

Output :

```
This is an example
```

What happens?

1. When the above program is executed, the editor will send all the lines in the program into a Python interpreter.
2. An interpreter will decipher all the words in the program and determine its purpose.
3. If the interpreter is confused or finds words that it cannot determine, it will throw out warnings, exceptions, and errors.
4. When the above example is executed, an interpreter will look at the 'print' statement and understand its purpose is to display whatever is there between parentheses on the screen.

Now, look at a slight variation of the above example.

Program code :

```
first = " This is an example"
print(first)
```

Output :
```
This is an example
```

In the above example, 'first' is used as a variable to store a string in a specific memory location. To be precise, the variable named 'first' now holds a value and can be called at any instance.

Python is not a statically typed language, and hence there is no need to declare variables while creating them. Programmers can also change the value of Python variables at any instant.

Program code :
```
first = " This is an example"
print(first)
first = " This is a second example"
print(first)
```
Output :
```
This is an example
This is a second example
```

The variable's value has changed instantaneously after declaring a different string for the variable in the above example.

HOW TO NAME VARIABLES

Python programming guidelines provide strict rules while creating variables, so they do not fall prey to errors. This also improves the program's readability.

Rules for writing variables:

1. Python only allows numbers, letters, and underscores for creating variable names. For example, '$message' is not a qualified variable name.
2. Python programmers should not start variable names with a number. For example, 'message1' is an allowed variable name, but '1message' is not allowed.
3. Python doesn't allow you to use 33 reserved keywords as identifiers while creating variables. For example, 'print' cannot be used as a variable name.
4. Always write variables that are easy to follow. Writing complex variable names is considered a bad programming practice, especially if you work with teams.

HOW TO DEFINE VARIABLES

You can define variables in a Python program using the assignment operator '='.

Syntax format :

```
Variable name = " Variable value"
```

Example :

```
first = 2
# This is a variable with integer data type
second = " United States Of America."
# This a variable with String data type
```

HOW TO FIND THE MEMORY ADDRESS OF THE VARIABLES

Finding the memory address of the variables is important, especially while working with complex applications that constantly have a necessity to change their values.

Traditional languages such as C use pointers for using memory addresses in programs.

Python provides a way to provide the memory address of the variables easily. Usually, in high-level programming languages such as C and C++, pointers are used to find the memory address. But in Python, we need to use a built-in function called id() to get the memory address of the variable.

Program code :
```
first = 324
id(first)
```
Output :
```
1x23238200
```

In the above example, 1x23238200 is the memory address of the variable 'first'. When a variable is replaced, the memory address will remain the same.

Program code :
```
first = 324
id(first)
first = 456
id(first)
```
Output :
```
1x23238200
1x23238200
```

LOCAL AND GLOBAL VARIABLES IN PYTHON

Python can have variables that can be declared to be used only inside a function. Local variables cannot be assigned or called outside the function.

Program code :

```
#This is a function with a local variable
def exa() :
d = " This is a local"
print(d)
exa()
```

Output :

```
This is a local
```

In the above example, we declared a variable with a local scope. On the other hand, global variables are usually declared outside a function but can still be used to call inside a function.

Program code :

```
def exa() :
print(d)
#This is a global scope variable
d = " This is a global"
exa()
```

Output :

```
This is a global
```

Programmers can use variables with different data types such as integers, strings, lists, and dictionaries. Variables can also be combined with operators for creating complex programs.

RESERVED KEYWORDS

Every programming language has its prebuilt syntax known as reserved keywords usually needed to facilitate the language. All these reserved keywords clearly define a specific functionality while building the programming language from scratch.

For example, "if" is a reserved keyword in Python and is used to create a conditional statement while writing Programs.

Programmers cannot use reserved keywords while creating identifiers for other programming components such as variables, functions, and classes. Python 3 currently has 33 reserved keywords.

Do it Yourself:

Open your Python terminal and enter the following commands to list out all reserved keywords.

Program code :
```
>>> import keyword
# This statement helps to import a library
>>> keyword.kwlist
# All the reserved keywords will be displayed
```

OPERATORS IN PYTHON

Operators are usually used in mathematics to combine literals and form statements.

Example :
```
2x + 3y = 7
```

Here, 2x, 3y, and 7 are literals, whereas + and = are operators. Computer scientists adopted operators from mathematics and used them effectively to assign and manipulate values.

Operators are designed for computation and hence can be combined with objects to create complex expressions. A group of expressions usually form an effective computational logic.

In Python, the values that an operator tries to combine are known as operands.

Example :
```
>>> first = 45
>>> second = 78
>>> first + second
```

Output :
```
133
```

In the above example, first and second are operands, whereas '+' is the operator.

DIFFERENT TYPES OF OPERATORS

Operators are usually complex and are of different types. The most popular ones are arithmetic operators, which are usually used to apply mathematical operations.

Any mathematical operation such as Addition, Subtraction, Multiplication, and Division can be applied to the variables and lists using these operators.

1. Addition (+)

The addition operator is usually used to add two literals; ‘+’ is its symbol. When you use the addition operator to add two different data types, the Python interpreter will automatically apply typecasting for suitable data types.

Program code :
```
A = 42
B = 33
C = A + B
# + is an addition operator
Print(c)
```
Output :
```
75
```

2. Subtraction

The subtraction operator is usually used to find the difference between two literal values or lists. ‘-’ is its symbol.

Program code :
```
A = 42
B = 33
C = A - B
# - is a subtraction operator
Print(c)
```
Output :
```
9
```

3. Multiplication

Multiplication operator is usually used to find the product between two literal values or lists. It is represented using the ‘*’ symbol.

Program code :
```
A = 42
B = 33
C = A * B
# - is a multiplication operator
Print(c)
```
Output :
```
1386
```

4. Division

The division operator is usually used to find the quotient when two numbers of any data type are divided. Whenever two numbers are divided, you will usually be given a floating-point number as a result. The symbol for the division operator is ‘/’.

Program code :
```
A = 42
```

```
B = 33
C = A \ B
# / is a division operator
Print(c)
```
Output :
```
1.272
```

5. Modulus

Using this special operator, you can find a number modulo with another number. Mathematically we can call modulo also as the remainder between two numbers. The symbol of the modulus operator is ‘%’.

Program code :

```
A = 42
B = 33
C = A % B
# % is the modulus operator
Print(c)
```

Output :

```
9
```

6. Floor division

Floor division is a special arithmetic operator that provides the nearest integer value as the quotient value instead of providing a floating-point number. The symbol of the floor division operator when writing Python programs is ‘//’.

Program code :

```
A = 42
B = 33
C = A // B
# // is a floor division operator
Print(c)
```

Output :

```
1
```

OPERATOR PRECEDENCE

When you create and work with mathematical expressions, it is also important to know when and how to perform a mathematical operation. For example, when there are multiple operators in an expression, it is very important to follow strict rules that exist to calculate them. If not, the value may completely change.

Python programmers can follow operator precedence rules to avoid these problems while creating software.

Rules:

1. parentheses is the element with ultimate precedence in a Python expression. If you find any literals or variables inside a parenthesis, then you need to calculate them first before proceeding to others.
2. The second precedence is exponent operator followed by bitwise operators
3. The next precedence order will be included with multiplication, division, modulus, and floor division operators.
4. The next precedence will be given to addition and subtraction operators
5. Any logical or comparison operators will be with the last precedence will working with the Python programs

4

DATA TYPES IN PYTHON

Python programming language deals with a lot of different types of data to create universal applications. Understanding these different sets of values, the operations that programmers can make, and their importance in software development are essential for beginners.

WHAT ARE DATA TYPES?

In simple terms, data types are a set of values that programmers define while creating variables in programming languages. As Python is not a statically typed language, there is no rule to define them specifically. The Python interpreter will usually automatically detect the data type for a variable and perform operations. Even though there is no need to define them, understanding data types is an essential skill for creating intricate programs.

For example, when you create a variable named 'first' with a value 10, you typically create a variable 'value' with a value '10' of data type 'integer'. Python automatically detects the data types of all the variable values as it is a dynamically typed language.

Program code :

```
first = 10
```

Statically typed languages such as C make it mandatory for programmers to mention the data type when declaring a variable.

For example, in C language, a variable with a float data type is declared, as shown below.

Program code :
```
float age = 64;
```

UNDERSTANDING DATA TYPES

To understand data types in much more depth, you need to understand some of the essential code fragments in a Python program.

Example :
```
a = 64
b = 54
c = a + b
```

Here a, b, and c act as variables and '+' and '=' are operators.

The value given to the variables is known as a literal and represents a data type value. 64 and 54 are the literals in the above example. A literal usually acts as a directive to insert a specific value into a variable.

The popular data types in Python are integer, float, string, and Boolean. Understanding data types in depth and the operations possible on them is a mandatory prerequisite for Python programmers.

Strings

Strings are data types that are used to represent text in programs. Strings are usually enclosed in single quotes and have a sequence of characters. When you use a string data type, an 'str' object with a sequence of characters is created.

String data types are necessary because a computer reads data in the form of binary. It is required to convert characters using an encoding mechanism so that the computer can understand and manipulate them according to our instructions. ASCII and Unicode are some of the popular encoding mechanisms programming languages use to read text data.

Until Python 2, there was no way to interact with foreign languages such as Chinese and Korean effectively. Python 3, however, has introduced a Unicode mechanism to deal with data other than English.

How Are They Represented?

Program code :

```
a = ' This is a string.'
print(a)
```

Output :

```
This is a string.
```

Everything in between the quotes can be called a string literal, and it can be called anywhere in the program using the variable 'a'. In the above example, the string consists of 14 characters, excluding whitespaces.

Python strings can also be represented using three other different ways.

Program code :
```
a = " This is a string."
# Literals in double quotes
print(a)
a = ''' This is a single line '''
# Literals in triple single quotes
print(b)
a = """ This is a multiple
lines program """
# Literals in triple double quotes for
representing multiple lines
print(c)
```
Output :
```
This is a string
This is a single line
This is a multiple lines program
```

String data types in Python can be used to create any amount of information. All special characters, symbols, letters, digits, and whitespaces can be entered between the quotes. Python also provides particular components known as escape sequences to format the data present effectively. For example, programmers can use '\n' to create a new line.

How to Access Characters in a String

Python provides different ways to interact with string data easily. We can easily access a specific location of a string using indexes. Python also uses the concept of slicing to access a range of characters in a string.

Positive indexing usually starts with 0, and negative indexing starts with = -1. For slicing, a colon (:) is used to provide the range.

Examples:

```
# Accessing Strings in Python
sample = ' Python'
print('sample =' , sample)
# Will print the first character
print(' sample[0] = ', sample[0])
# Will print the last character
print(' sample[5] = ', sample[5])
# Will slice from 3rd to 6th character
print(' sample[2: 5] = ', sample[2:5])
# Using negative indexing
print(' sample[-1] = ', sample[-1])
```

Output :

```
sample = Python
sample[0] = P
sample [5] = n
sample [2:5] = thon
sample [-1] = n
```

Because they are immutable, it is impossible to either change or delete elements in a string. You can, however, reassign the string literals easily. When you try to replace a string element using indexing, a type error will occur.

String Formatting

Strings are usually used to display the information that wants to appear when action appears in the program. Many also use strings to display input values dynamically given by the user.

To perform string formatting, Python programmers can primarily follow two different strategies.

1. Format with placeholders

In this method, programmers are required to use the % (modulus) operator to format the appearance of the strings. Using placeholders to format strings is the oldest method, and '%' is famously called a string formatting operator.

Program code :
```
>>> print ( " %s is a great game" %
'football')
```
Output :
```
football is a great game
```

Using the placeholders method, you can also format two strings at the same time.

Program code :
```
>>> print ( " %s is the national sport of %s"
('Baseball', 'USA') )
```
Output :
```
Baseball is the national sport of USA
```

Remember that while %s is used to format strings, you can use %d to format integers and %f to format floating-point numbers.

Program code :

```
>>> print ( " Brazil won FIFA world cup %d
times" % 5)
```

Output :

```
Brazil won FIFA world cup 5 times
```

To format floating-point numbers in a string, you need to use a different format known as %a.bf. Here, a represents the minimum number of strings displayed on the screen, and b represents the number of digits used after the example.

Program code :

```
>>> print ( " The exchange rate of USD to EUR
is : %5.2f" % (0.7546))
```

Output :

```
The exchange rate of USD to EUR is : 0.75
```

2. Format with .format() string

Python 3 has provided a special method known as .format() to format strings instead of the traditional modulus (%) operator. Python 2 also has recently adopted the .format() method into its standard library.

Example :
```
>>> print ( “ Football is popular sport in
{}.” .format(' South America') )
```
Output :
```
Football is popular sport in South America
Using .format(), we can also format the
strings using indexed based positions.
```
Program code :
```
>>> print( ' {3} {2} {1} {0}' .format ( '
again' , 'great' , ' America' , ' Make') )
```
Output :
```
Make America great again
```

Apart from these two ways, you can also format strings in Python using f-strings. Lambda expressions can be used in this method and are used when there is a requirement for advanced formatting scenarios.

While it is tempting to use the placeholder method as it is easy to implement, we recommend using the .format() method as it is considered the best practice by many Python 3 experts. The placeholder method usually takes a lot more time to execute during runtime and is not a good solution, especially for large projects.

String Manipulation Techniques

Strings are the most utilized data type in programs because data instantiation usually happens through strings. Understanding string manipulation techniques is a prerequisite skill for Python programmers. Many data scientists and data analysts solely depend on these manipulation techniques to create rigorous machine learning models.

The most important string manipulation technique Python programmers need to be aware of is concatenating and multiplying the strings.

1. Concatenate

Concatenate refers to joining strings together to make a new string. To concatenate two strings, you can use the arithmetic operator '+'. You can also separate the added strings using white space for better readability.

Program code :

```
Sample = ‘ This is ‘ + ‘ ‘ + ‘ FIFA world
cup’
Print( sample)
```

Output :

```
This is FIFA world cup
```

In the above example, if the white space is not provided, both strings will be together as Python interpreter usually displays output without formatting.

2. Multiply

Multiplying refers to copying strings again and again easily. A Python programmer can use the mathematical operator "*" to multiply strings easily.

Program code :

```
Sample = ‘ welcome ‘ * 4
# This makes the string to multiply itself 4
times
```

```
Sample1 = ‘ To whoever you are ‘
Print( sample + sample1)
```
Output :
```
welcome welcome welcome welcome To whomever
you are
```

In the above example, we first multiplied the strings and then concatenated them to another string.

3. Appending strings

While both concatenating and multiplying strings are considered basic string manipulation techniques, you can further improve your strings efficiency by appending the strings using a special operator (+=).

Example :
```
Sample = ‘ This is a great’
Sample += ‘example’
Print(sample)
```
Output :
```
This is a great example
```

In the above example, += is used to append another string at the end. You can, however, not use this operator to add information in the middle of the string.

4. Length

Apart from using string operations, you can use certain prebuilt string functions in the Python library to do additional tasks. Having a good grasp of these techniques can help programmers to format their data in Python applications effectively.

Length is one of these methods that help programmers to mention the number of characters in a string. Remember that blank space will also be added as a separate character in the count.

Program code :

```
Sample = ‘ Today is Monday’
Print(len(sample)
```

Output :

```
13
```

5. Find

Finding a part of the substring in a large string is an important task for many Python programmers. Python provides an inbuilt find() function for this purpose. This method will output the starting index position of the substring.

Note:

Python programmers can use only positive indexes, and also the index starts with 0.

Example :

```
Sample = ‘ This month is August’
Result = sample. Find(“ Augu”)
Print(result)
```

Output :
```
12
```

If you try to find a substring that is not present in the main string, then the Python interpreter will give the result as ‘-1’

Program code :
```
Sample = ‘ This month is August’
Result = sample. Find(“ July”)
Print(result)
```
Output :
```
-1
```

6. Lower and higher case

You can use the .lower() method to convert all the characters in a string to lowercase letters.

Program code :
```
Sample = “ Football is a great sport”
Result = sample. Lower()
Print(result)
```
Output :
```
football is a great sport
```

In the above example, if you notice, you can find that all the uppercase letters are converted into lowercase.

You can perform a similar operation with the help of the .upper() method, but to convert all the characters in a string to upper case letters.

Program code :
```
Sample = “ Football is a great sport”
```

```
Result = sample. Higher()
Print(result)
```
Output :
```
FOOTBALL IS A HIGHER SPORT
```

7. title()

A Python programmer can also easily use the title() method to convert a string to camel case format.

Program code :
```
Sample = “ Football is a great sport”
Result = sample. Title()
Print(result)
```
Output :
```
Football Is a Great Sport
```

8. Replace strings

A Python programmer often needs to replace a part of the string. You can achieve this using the inbuilt replace () method. However, remember that all the characters in the string will be replaced whenever you use this method.

Program code :
```
Sample = “ Football is a great sport”
Result = sample.replace( “ Football”, “
Cricket”)
Print(result)
```
Output :
```
Cricket is a great sport
```

Apart from all these manipulation techniques, you can also use escape sequences such as print line(\l) and new tab (\t) to format the data on the output screen effectively.

Integers

Integers are the data types in Python that are used to represent numbers from 0 to 9. Programs often use ‘int’ literals because they are required for both program logic execution and complex arithmetic calculations.

When a Python interpreter encounters an ‘int’ while execution, it creates an int object with a specified value. This value can be replaced whenever we need to.

"int" objects are often used in programs and are the basic building blocks of an efficient program. For example, colors on a web page use a string of values that are represented in integers.

Python also uses operators such as +,-,*,/ on these literals. As a beginner, you need to remember the unary operators ‘+’ and ‘-’, representing the sign of an integer.

For example, +12 represents a positive integer, whereas -12 represents a negative integer.

Representing Integers

```
a = 6
print(a)
```

Output :

```
6
```

It also should be remembered that the range of integer literals is usually huge in Python. You can input literals with more than ten digits in a variable. Even though there is no problem with using large integers, there is always a problem of memory leakage when a program fills the computer's memory with operations involving huge integers. To get rid of bottleneck situations, make sure that expressions involving large integer literals are practical.

Floating-Point Numbers

Float is a special data type in Python that programmers can use to represent floating-point numbers while working with program variables. Usually, when we create software, we need to work with real numbers along with a decimal point. You can represent as far as ten decimal points using a 'float' data type variable.

Program code :
```
A = 7.1213
Print(a)
# This represents a floating-point variable
```
Output :
```
7.1213
```

You can also use a floating-point variable to represent a hexadecimal integer, a popular numerical representation method.

Example :
```
A = float.hex ( 3.1212)
Print(a)
```
Output :
```
'0x367274872489'
```

Python programmers can also use floating-point numbers to represent both complex and exponential numbers.

Boolean Data Type

Boolean is a special data type that can help programmers to represent true or false during an instance. The two Boolean values are TRUE and FALSE.

You can mostly use Boolean data types along with relational operators.

Example:
```
Print ( 100 < 32)
```
Output :
```
False
```

As this is a false statement, the Python interpreter will output Boolean value ‘False’ to the programmer.

TYPE CASTING

Usually, when we work with programs, situations occur when the data type of a variable needs to be changed. Python provides an opportunity to convert data types without messing up the program using casting.

Usually, type casting can be performed using two types: implicit and explicit type casting.

1. Implicit type casting

Using implicit type casting, Python automatically converts the data types whenever required.

Example:
```
X = 32
Print(type(x))
# This prints the data type of x
Y = 2.0
Print(type(y)
# This prints the data type of y
Z = x +y
Print(Z)
Print(type(Z))
# implicit type casting occurs
```
Output :
```
<class ‘int’>
```

```
<class ‘float’>
64.0
<class ‘float’>
```

In the above example, when a float variable is multiplied with an integer variable, the result has already been implicitly converted to a floating-point data type variable.

2. Explicit type casting

When a Python programmer changes the variable data types with their involvement, it is known as explicit type casting. Usually, explicit type casting is performed using data type functions.

Typecasting int to float :

```
X = 32
# This is an int variable with a value ‘32’
Z = float(x)
# Now the int data type is type casted to
float
Print(Z)
Print(type(Z))
```

Output :

```
32.0
<int ‘float’>
```

In the above example, we have converted a variable with data type ‘int’ to data type ‘float’ using implicit type casting.

Typecasting float to int :

```
X = 32.0
# This is an int variable with a value ‘32’
Z = int(x)
```

```
# Now the float data type is type casted to
int
Print(Z)
Print(type(Z))
```
Output :
```
32
<class ‘int’>
```

In the above example, we have converted a variable with data type ‘float’ to data type ‘int’ using implicit type casting.

Typecasting int to string :

```
X = 32
# This is an int variable with a value ‘32’
Z = str(x)
# Now the int data type is type casted to
float
Print(Z)
Print(type(Z))
```
Output :
```
32
<int ‘string’>
```

In the above example, we have converted a variable with data type ‘int’ to data type ‘string’ using implicit type casting.

Typecasting string to int :

```
X = 32
# This is a string variable with a value ‘32’
Z = int(x)
# Now the string data type is type casted to
int
```

```
Print(Z)
Print(type(Z))
```
Output :
```
32
<class ‘int’>
```

In the above example, we have converted a variable with data type ‘int’ to data type ‘float’ using implicit type casting.

Exercise:

Following the examples mentioned earlier, try to type cast a string variable to a float variable using the same input data.

5

LISTS AND TUPLES

As a programmer, you need to deal with a lot of data instead of single linear data. Python uses data structures such as lists, tuples, and dictionaries to handle more data efficiently. Understanding these data structures that help Python programmers manipulate and easily modify the data is an essential prerequisite for beginners.

WHAT IS A LIST DATA TYPE?

A list is a unique data structure that can hold multiple values of different data types in sequential order. Whenever programmers mention ‘list value,’ it represents the whole list itself. Like any variable, the list itself can be manipulated, replaced, passed to a function and can destroy the Python variable mechanism.

Lists are usually represented as shown below:

[64,48,32]

Here 64, 48, and 32 are list elements. All the elements in the list are of integer data type.

Lists usually start with a square bracket and end with a square bracket. Commas will usually separate all the elements in the list. If the elements in the lists are of string data type, they will be enclosed within quotes. Elements inside the list can also be called items.

Example :
```
[USA, China, Russia]
Here USA, China, and Russia are the elements
in the list. They are string data types.
List to variable:
>>> sample = [USA, China, Russia]
> > > sample
```
Output:
```
[USA, China, Russia]
```

In the above example, the variable ‘sample’ has now stored the list provided. You can call the list using a variable like you print any other value of standard data types on the screen.

Empty List

If the list doesn’t consist of any elements, it is called an empty or null list.

It is usually represented as [].

Program code :
```
>>> sample = [ ]
# This is an empty list
```

Understanding Index in Lists

Every element in the list can be individually called, manipulated, or modified using a unique Python feature known as indexes. Usually, the index numbers start with ‘0’, and you can use them as shown in the below example.

Let us suppose that the list is [USA, China, Russia].

Program code :

```
>>> sample = [USA, China, Russia]
>>> sample[0]
>>> USA
>>> Sample[1]
>>> China
>>> sample[2]
>>> Russia
```

In the above example, when we insert the index number ‘0’, the interpreter will print the first element in the list. Similarly, we can call any element in the list with its respective index.

Example :

```
>>> ' Hi’ + sample2
```

Output:

```
Hi Russia
```

You can also call an element in your list in the following manner.

Program code :

```
>>> [USA, China, Russia] [1]
```

Output :

```
China
```

If you provide an index value more than the maximum index, then the Python interpreter will throw an error.

Program code :

```
Sample = [USA, China, Russia]
```

```
Sample[3]
```
Output :
```
Index Error: list index out of range
```

Note : You can only use integer data type for indexes. Usage of floating-point data type can result in TypeError.

Program code :
```
>>> sample = [USA, China, Russia] [1.0]
```
Output :
```
TypeError: list index should not be float but
should be an integer
```

Remember that all the list elements can consist of other lists in them. To be precise, all the parent lists can have child lists.

Program code :
```
>>> sample = [[Football, Cricket, Baseball],
USA, China, Russia]
>>> sample[0]
```
Output :
```
[Football, Cricket, Baseball]
```

You can also use child list elements using the following index format.

Program code :
```
>>> sample[0][2]
```
Output :
```
Baseball
```

In the above example, the interpreter has first used the first list in the element and then later for the third element to print it on the screen.

You can also call list elements using negative indices. Usually, -1 refers to the last element and -2 to the last before.

Program code :
```
sample = [USA, China, Russia]
Sample[-1]
# will print the last element
```
Output :
```
'Russia’
```

Slicing Using Lists

You can create sub-lists easily using the slicing method. Using the slicing method, you can extract specific elements and create separate new lists.

Syntax :
```
Listname index start : index end
```

A colon usually separates the index numbers. The first index number will provide the value about where the slicing should start, and the end index number will provide the value about where the slicing should end.

Program code :
```
>>> sample = [USA, China, Russia, Japan,
South Korea]
>>> sample[0 : 2]
```
Output :
```
[USA, China, Russia]
```
Program code :
```
>>> sample[2 : 3]
```
Output :

```
[Russia, Japan]
```

You can also slice without entering the first or last index number. For example, if the first index number is not provided, the interpreter will start from the first element in the list. In the same way, if the last index number is not provided, then the interpreter will extend the slicing till the last element.

Program code :
```
>>> sample = [USA, China, Russia, Japan,
South Korea]
>>> sample[ : 1]
```
Output :
```
[USA, China]
```
Program code :
```
>>> sample[2 :]
```
Output :
```
[Russia, Japan, South Korea]
```

If you don’t provide both values, then the whole list will be given as an output.

Program code :
```
>>> sample [ : ]
```
Output :
```
[USA, China, Russia, Japan, South Korea]
```

Getting List Length

The inbuilt lens() function can help a Python programmer print out the list's length.

Program code :

```
>>> sample = [ USA, China, Russia, Japan,
South Korea]
> > > len(sample)
```

Output :

```
5
```

You can directly change values inside lists using the assignment operator. Remember that whenever you replace them, the value will be lost forever, and you cannot retrieve them back. Only let your programs use them if it is necessary.

Program code :
```
>>> sample = [USA, China, Russia, Japan,
South Korea]
>>> sample [2]= [‘Germany’]
>>> print(sample)
```
Output :
```
[USA, China, Germany, Japan, South Korea]
```

In the above example, after using the assignment statement on the second element, the value has changed from ‘Russia’ to ‘ Germany’.

You can also directly replace one list value with another using the assignment statement.

Program code :
```
>>> sample = [ USA, China, Russia, Japan,
South Korea]
>>> sample[2] = sample[4]
```
Output :
```
[USA, China, Russia, Japan, Russia]
```

In the above example, the end element in the list is replaced by the second element in the list.

List Concatenation

You can add two different lists using the "+" operator. However, remember that both should be of the same data types. If not, then it may throw errors sometimes.

Program code :
```
[4,5,6] + [5,6,7]
[4,5,6,5,6,7]
```

List Replication

You can multiply two strings using the list replication technique. To perform the replication technique, you need to use the "* "operator.

Program code :
```
[4,5,6] * 3
```
Output :
```
[ 4,5,6,4,5,6,4,5,6]
```

You can also easily remove an element from a list using del statements. However, remember that when you delete a middle element, the value of indices will change automatically, so your results may change. Confirm whether or not it will affect any other operation before proceeding with the deletion.

Program code :
```
>>> sample = [USA, China, Russia, Japan,
South Korea]
>>> del sample[3]
>>> sample
```
Output :
```
[USA, China, Russia, South Korea]
```

'in' and 'not in' Operators for Lists

Python provides two operators called 'in' and 'not in' to determine whether or not an element is there in a list. All the output results for these operators will be given as a Boolean value.

Program code :
```
>>> 'China' in [USA, China, Russia, Japan,
South Korea]
```
Output :
```
TRUE
```
Program code :
```
Sample = [USA, China, Russia, Japan, South
Korea]
'Mexico' not in sample
```
Output :
```
TRUE
```

You can also use multiple assignment statements to allot a list element to a variable.

Example :
```
Sample = [USA, China, Russia, Japan, South
Korea]
Result1 = sample[0]
Result2 = sample[2]
```

```
Result3 = sample[1]
```

Instead of assigning a variable to an individual list item separately, you can use the following format.

Program code:

```
Result1,result2,result3 = sample
```

However, remember that you can only use this with the exact elements you have on your list. Otherwise, a type error will occur and will crash the program.

Finding Value In a List Using The index() Method

Python also provides various inbuilt functions that help programmers to extract different additional data from these data structures.

Program code :
```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.index(‘Japan’)
```
Output :
```
3
```

In the above example, the index method has mentioned a list element as an attribute, and so it gave the index number of the list as a result. If you provide a list element that doesn’t exist, then a type error will occur.

Program code :
```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.index(‘Switzerland ’)
```
Output :
```
Type error : list element doesn’t exist
```

Adding Values to a List Using append() And insert()

Adding list elements in your favorable index position is essential when you are working with python programs.

Program code :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.append(‘ Canada’)
Sample
```

Output :

```
[USA, China, Russia, Japan, South Korea,
Canada]
```

The above example used the append() method to add an element to the end of the list.

You can also use the insert() method to insert a list element at any index position. Insert method also uses a specific format that asks the programmer to insert the index number where the element should be inserted.

Insert (index position, ‘item’)

Example :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.insert(3, ‘Canada’)
Sample
```

Output :

```
[USA, China, Russia, Canada, Japan, South
Korea]
```

In the above example, a new element called ‘Canada’ is added in the index position 3, and the previously present elements move an index forward.

Removing List Elements Using remove()

Using the remove() method, a programmer can quickly delete an element from the list.

Program code :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.remove(‘Japan’)
Sample
```

Output :

```
[USA, China, Russia, South Korea]
```

In the above example, the element ‘Japan’ is removed from the list using the remove() method. If you try to remove an element that doesn’t exist in the list, a value error will occur.

Program code :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.remove(‘Canada ’)
Sample
```

Output :

```
Value error : The list element doesn’t exist
```

Sort List Elements Using sort()

Sorting is an essential trick to arrange numbers or string values present in an order. Usually, when we use the sort() method, all the elements will be arranged in ascending order.

Program code :

```
Sample = [3,5,1,9,4]
Sample.sort()
```

Output :

```
[1,3,45,9]
```

You can also sort string elements. The elements will be ordered using alphabetical order.

Program code :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.sort()
Sample
```

Output :

```
[China, Japan, Russia, South Korea, USA]
```

To sort the elements in descending order, you can use ‘reverse’ as a keyword argument along with the sort() function.

Program code :

```
Sample = [USA, China, Russia, Japan, South
Korea]
Sample.sort(reverse = True)
Sample
```

Output :

```
USA, South Korea, Russia, Japan, China
```

Remember that when you try to sort elements in a list, all the elements in the list should be of the same data type. If not, the Python interpreter, which has no way to compare values between two different data types, will throw a TypeError.

Program code :

```
Sample = USA, China, 3,4,5.4
sample.sort()
Sample
```

Output :

```
TypeError: These values cannot be compared
```

UNDERSTANDING TUPLES

Lists are mutual objects in Python. All the elements in a list can be easily removed, added or deleted using different functions. Tuples, on the other hand, are immutable lists. Any element that is entered into a tuple cannot be modified. Trying to change an element in a tuples data structure can result in a “TypeError”.

Also, remember that they are denoted using parentheses instead of square brackets that list data structures. You can also represent tuples without using parentheses.

Program code:
```
Sample = (‘football’ , ‘baseball’ ,
‘cricket’)
Print(sample)
# creating a tuples without a parenthesis
```
Output:
```
‘football’, ‘baseball’, ‘cricket’
```
Program code:
```
Sample = (‘football’ , ‘baseball’ ,
‘cricket’)
Print(sample)
# creating a tuples with a parenthesis
```
Output:
```
(‘football’, ‘baseball’, ‘cricket’)
```

Concatenating Tuples

Just like lists, you can add or multiply tuple elements in Python.

Program code:
```
Sample1 = (21,221,22112,32)
Sample2 = (‘element’, ‘addition’)
```

```
Print( sample1 + sample2)
# This concatenates two tuples
```
Output:
```
(21,221,22112,32,’element’,‘addition’)
```

You can also nest a tuple into another using Python.

Program code:
```
Sample1 = (21,221,22112,32)
Sample2 = (‘element’, ‘addition’)
Sample3 = (sample1,sample2)
Print(sample3)
```
Output:
```
( ( 21,221,2212,32), (‘element’, ‘addition’))
```

You can also repeat the elements in a tuple using the replication technique.

Program code:
```
Sample1 = (‘sport’, ‘games’) * 3
Print(sample1)
```
Output:
```
(‘Sport’,’ games’,’ sport’, ‘games’, ‘sport’,
‘ games’)
```

Remember that tuples are immutable. So, if you try to change a value of the element, then a TypeError will occur.

Program code:
```
Sample1 = (21,221,22112,32)
Sample1[2] = 321
Print(sample1)
```
Output:
```
TypeError: Tuple cannot have an assignment
statement
```

How to Perform Slicing in Tuples

To perform slicing, you need to enter the starting and ending index of tuples divided by a colon (:).

Program code:
```
Sample1 = (21,221,22112,32,64)
Print(sample1[2 : 4])
```
Output:
```
(22112,32,64)
```

How to Delete a Tuple

Even though you cannot delete elements present in the tuple separately, you can delete the tuple itself.

Program code:
```
Sample1 = (21,221,22112,32,64)
Del sample1
```

Now, when you call the tuple, it will receive a TypeError because the tuple has already been deleted.

UNDERSTANDING DICTIONARIES

Dictionary is a special data structure that helps Python store values as a map instead of a single value data element. Dictionaries use the “key : value” pair to make the data more optimized. Dictionaries use curly brackets to represent them in a program.

How to Create a Dictionary

To create a dictionary, you need to place a sequence of elements in curly brackets. Dictionaries usually hold a pair of values and are separated by a comma.

Syntax:
```
Dictionary = [key : value , key : value]
```

You can insert any value such as a variable or a list in these key : value pairs.

Example :
```
Sample = {1 : USA , 2 : China , 3 : Brazil ,
4 : Argentina}
Print(Sample)
```
Output:
```
{1: USA, 2: China, 3: Brazil, 4: Argentina}
```

You can also create a nested dictionary using Python.

Program code:
```
{1 : USA , 2 : China , 3 : Brazil , 4 : { 1 :
Football , 2 : Cricket , 3 : Baseball } }
```

To add an element to a dictionary, you can use various techniques. The most popular one is adding all the elements.

SAMPLE EXERCISE

Write a Python program to print Even Numbers in a List.

Program code:

```
#These are sample numbers we have used
sample = [22,32,11,53,98]
# Using a for loop to solve the logic
for num in list1:
# Logic for detecting even numbers
if num % 2 == 0:
print(num, end = " ")
```
Output:

```
22,32,98
```

EXERCISE:

1. Write a Python program that can create a matrix using lists and provide an inverse matrix.
2. Write a Python program to form a few lists and interact with each other to play a word scrambling game.
3. Write a Python program to reverse all elements in the list and find out the character length of all the strings in the list
4. Write a Python program to effectively ascend or descend the values and key Pair present in them
5. Write a Python program to create a list that provides popular 100 English synonyms with meanings
6. Write a Python program to invert a dictionary and replace their elements with the RGB values of Blue, Green, and Orange colors.

6

USER INPUT AND OUTPUT

Most of the programs are designed to provide a purpose for the user. Most of the software that became popular provides a unique solution to the user. To provide a unique experience to software, you need to get some information from the user. Python programmers need to access this user input and perform operations based on that input, usually for providing a unique experience to the user. All real-world applications need both user input and output for working efficiently.

With the help of the information provided about input and output in this chapter, you will be able to create long programs that perform according to user’s input.

WHY ARE INPUT VALUES NECESSARY?

Software behaving based on input provided by users is one of the general rules to create great software. When you log in to Facebook, you will input your email and password, and the database will verify your credentials based on the input data. Even facial recognition technology uses your face mapping as input data. Every real-world application asks the user to provide input in one way or another.

For example, let us suppose that you are supposed to let users access the application only if they are 18 years old. To implement this condition, you have created a programming logic that verifies users' age and passes them into the application only if they are 18. However, for this programming logic to apply, the user first needs to enter their age using the input functionality. Inputs can be of any data type that Python supports.

UNDERSTANDING INPUT() FUNCTION

Whenever you use an input() function while writing Python programs, it will pause the program during execution and wait for the user to enter an input depending on the prompt provided. Once entered, the user usually needs to press ‘enter’ to store the provided input inside a variable.

Example :

```
Sample = input ( “ Do you like football or
baseball more? “)
Print(sample)
```

When the above example is executed, the user will first see an output as shown below.

Output:
```
Do you like football or baseball more? :
Football
```

You should now enter a string. For example, let us suppose that you have entered football as an input. The output will then be shown as the program asked to print the input for the user.

Output:
```
Football
```

When you use an input() function, your only argument is the prompt that will be displayed on the screen.

HOW TO WRITE CLEAR PROMPTS FOR THE USER

Whenever you use the input() function, you must create a prompt that users can easily understand and respond accordingly.

Your prompt should be in simple words and dictate what you want the user to enter in the terminal. There should not be any fluff or unnecessary text.

Program code:
```
Sample = input( “ What is your nationality? :
“ )
Print ( “ So you are from “ + sample +” ! “)
```
Output:
```
What is your nationality? France
So you are from France!
```

You can also use the input() function to print multi-line strings as an input, as shown below.

Program code:

```
Prompt = “ This is a great way to know you “
Prompt += “ \n now say what is your age? "
Sample = input(prompt)
Print( “ \n You are “ + sample + “ years old”
)
```

Output:

```
This is a great way to know you
now say what is your age? : 25
You are 25 years old
```

INT() TO ACCEPT NUMERIC INPUT

Usually, when you obtain input from the user using the input() function, your input result will be stored in a string variable. You can verify it using an example.

If your result is enclosed in a single quote, then it is a string variable.

Storing values in strings is feasible only until you don’t need it for any other further calculations.

Example :

```
Sample = input ( “ What is your age? “)
```

Output:

```
What is your age? 25
```

However, now when we call it using a comparison operator, it throws back a TypeError.

Program code:

```
>>> sample >= 32
```

It throws an error because a string cannot be in any way compared to an integer.

To solve this problem, we can use an int() function that helps the interpreter enter details into an integer value.

Program code:

```
Sample = input ( “ What is your age? “)
```

Output:

```
What is your age? 25
```

Program code:

```
>>> sample = int(sample)
> > > sample >= 32
```

This will now display the output as FALSE as 25 is not greater than or equal to 32.

OUTPUT

Right from the beginning of this book, we have been using the print() function to output the results of a program on the screen. All the expressions you enter into a print() function will be converted into a string and displayed on the screen.

While most Python programmers usually use the print() function without any arguments, you need to be aware of different arguments that can change how you can output your data according to your use case and convenience.

Let us first look at a typical example with a print() function.

Program code:

```
print( “ This is how the print statement
works “ )
```

All the statement did is recognize the string in between parentheses and display it on the screen. This is how a print statement works usually. You can now learn about some of the arguments that the print() function supports in Python.

1. String literals

Using string literals in Python, you can format the output data on the screen. \n is the popular string literal that Python programmers can use to create a new blank line.

You can use other string literals such as \t, \b to work on other special instances to format the output data.

Program code:
```
Print ( " this is \n a great line " )
```
Output:
```
This is
a great line
```

2. End = “ “ Statement

Using this argument, you can add anything specified on the end once a print() function is executed.

Program code:
```
Print ( “this is a great line”, end =
“great”)
```
Output:
```
This is a great line great
```

2. Separator

The print() function can be used to output statements on the screen using different positional arguments. All these positional arguments can be divided using a separator. You can use these separators to create different statements using a single print() function.

Program code:
```
>>> Print( “ This is “ , “ Great”)
```
Output:
```
This is Great
```

SAMPLE EXERCISE

Write a program to accept floating numbers in a list as an input.

Program code:
```
list = []
x = int(input(" What is the list size? : "))
for i in range(0, x):
print("What is the location?", i, ":")
result = float(input())
list.append(item)
print("Result is ", result)
```
Output:
```
What is the list size? 5
2.3,5.5,6.4,7.78,3.23
Result is : 2.3,5.5,6.4,7.78,3.23
```

EXERCISES:

1. Write a Python program to get input from the user. Using this input, use different arithmetic all operators such as multiplication and division. You can also try to find the remainder.
2. Create a Python print() statement with a poem of your own choice.

3. Create a Python program that encourages Unicode developers to write code with good functionality.
4. Write a Python program to convert a decimal number to a hexadecimal number.
5. Write a Python program that defines both.

7

CONDITIONAL AND LOOPS

We already know that programs are a combination of a set of instructions. To make programs work effectively, we need to have the ability to skip or loop the instructions instead of executing them one by one, like a static list. Programming is dynamic, and to achieve several use cases, we need to understand the ability to control flow statements that all high-level programming languages support.

Without control flow statements, programs will become dumb and mechanical, increasing the program's runtime. A good programmer should always use programming components that can decrease program execution time. Programming languages provide control flow statements for this exact purpose. Understanding control flow statements is a prerequisite for advanced topics such as functions and object-oriented programming concepts that we will explore later.

COMPARISON OPERATORS

Understanding different comparison operators is an essential prerequisite for understanding the power that control flow statements can provide to a Python programmer.

Comparison operators, also known as relational operators, usually compare values present in two operands and return whether this condition is true or false using a Boolean value. There are different variations of these comparison operators in the core Python library.

'TRUE’ and ‘FALSE’ are the Boolean values that Python will display after successfully verifying the condition.

1. Less than (<) operator

Less than (<) operator usually checks whether or not the left value is lesser than the right value in the condition.

Program Code:
```
34 < 45
```
Output:
```
TRUE
```
Program Code:
```
45 < 34
```
Output:
```
FALSE
```

In the first example, the left value is lesser than the right value and hence ‘TRUE’ Boolean value is displayed as an output. However, the second example does not meet the condition, and hence ‘FALSE’ Boolean value is displayed on the terminal.

You can also compare int values with floating-point values using a less than operator.

Program Code:
```
5.2 < 6
```

Output:
```
TRUE
```

Programmers can use less than operator to compare strings using the ASCII format.

Program Code:
```
'Python’ < ‘python’
```
Output:
```
TRUE
```

In the above example, the statement has a ‘TRUE’ Boolean value because the lower case letters have higher ASCII values than the upper case letters.

Comparison operators can also be applied to tuples, as shown below.

Program Code:
```
(3,5,7) < (3,5,7,9)
```
Output:
```
TRUE
```

However, remember that when comparing with tuples, the data types should be the same. If not, a traceback error will occur and will crash the program.

Program Code:
```
(3,5,7) < (‘one’, 5,6)
```
Output:
```
Error: Cannot be compared
```

2. Greater than (>) operator

Greater than (>) operator usually checks whether or not the left value is greater than the right value in the condition.

Program Code:
```
34 >45
```
Output:
```
FALSE
```
Program Code:
```
45 > 34
```
Output:
```
TRUE
```

In the first example, the left value is not greater than the right value and the 'FALSE' Boolean value is displayed as an output. However, the second example met the condition, and the 'TRUE' Boolean value is displayed on the terminal.

Exercise:

Find out all different types of instances that we have tried for less than operator on greater than operator on your own.

3. equal (== operator)

We use this operator when we want to check whether or not two values are equal.

Program Code:
```
34 == 45
```
Output:
```
FALSE
```
Program Code:
```
23 == 23
```
Output:
```
TRUE
```

WHAT ARE CONTROL FLOW STATEMENTS IN PROGRAMS?

The program design of an application should follow a critical technique known as ‘control flow’ to communicate with users effectively. Using these statements, a user can perform actions necessary and ignore actions that are not.

Control flow statements are essential for every programming language. Without control flow statements, all the programs will be executed linearly, making them dumb and impractical. Conditionals and loops are the famous control flow statements that exist in the Python language.

PROGRAMMING STRUCTURES

Computer programmers use programming structures to represent logical reasoning to code effectively. Python programmers commonly use three programming structures to organize and execute code effectively.

1. Sequential structure

In a sequential structure, all the steps involved in a program are executed linearly.

Example :
```
A = 6
Print(a + “is a number” )
```
Output:
```
6 is a number
```

In the above example, the Python interpreter has executed all the steps linearly.

2. Conditional structure

The conditional structure is also called a selection structure and uses conditional statements to choose which steps of a program to execute. In this structure, only partial statements are executed as when a program makes a decision, the Python interpreter will ignore the other part in the conditional statement.

“If” and “if-else” are some of the famous conditional structure statements.

3. Looping structure

The looping structure uses conditions to execute the same programming step repeatedly until the condition is not satisfied. Loop termination is decided by the judgment condition that the programmer writes.

“While” and “for” loops are some of the famous looping structure statements.

IF/ELSE CONDITIONAL STATEMENTS

Decision making is essential while writing Python programs as all real-world applications need certain conditions to be satisfied for performing specific operations.

Python provides an if/else statement to make programmers implement the logic of decision making in their programs.

Syntax:
```
If Condition
Body Statement
Else
Body Statement
```

Program code:
```
Sample = 64
```

```
If sample % 4 == 0
Print ( “ This is divided by four “)
else :
Print ( “ This is not divided by four “)
```
Output:
```
This is divided by four
```

IF ELIF ELSE

You can further expand the ability of your programs by using multiple conditional expressions. If you have multiple statements that need to be checked, then using this conditional statement is recommended.

Program code:
```
Sample = 64
If sample > 0 :
Print ( “ This is a positive real number “)
Elif sample == 0 :
Print( “ The number is zero “)
else :
Print ( “ This is a negative real number “)
```
Output:
```
This is a positive real number
```

You can also create nested if/else statements to check more conditions and execute statements according to them.

FOR LOOP

For loop is used to iterate over some of the sequences repeatedly until they satisfy the condition. You can perform for loops with sequences such as lists, tuples, and strings.

Syntax:
```
For val in list :
Body of loop
```

The for loop usually goes through all the elements in the list until it is said so.

Example:

```
Sample = [2,4,6,8,10]
Result = 0
For val in sample :
Result = result + val
Print ( “ The sum of the numbers is “,
result)
```

Output:

```
The sum of the numbers is 30
```

In the above example, the for loop has executed all the numbers in the list until the condition is satisfied.

WHILE LOOP

While loop differs from for loop slightly. It can loop over any piece of code again and again until the condition is satisfied. The for loop is used when a programmer knows how many times a loop is going to be done, whereas the while loop is used when a programmer is not aware of the number of times a loop can happen.

Syntax:

```
While condition
Body statements of while
```

Example:

```
Sample = 10
First = 0
Second = 1
While second <= sample
First = first + second
Second = second + 1
Print ( “ The sum of numbers is: ", first )
```

Output:

```
Enter n: 4
```

```
   The sum of numbers is 10
```

You can interlink while and for loops with conditional statements to create complex programming logic.

BREAK AND CONTINUE STATEMENTS

Loops can be controlled and customized with the help of break and continue statements provided by Python. While loops are designed to iterate over a code block until they satisfy a condition, programmers can customize them to end or skip certain statements based on a result using break and continue.

What is Python break?

Whenever the Python interpreter sees a break in a loop statement, it ends the program flow. If a break is encountered in a loop, then the innermost loop will be terminated.

Syntax:
```
Break
```

What is Python continue?

Whenever the Python interpreter encounters a continue statement, it will skip the code that follows. The loop, however, doesn’t exit completely in this condition. It just skips to the next iteration once it encounters it.

Syntax:
```
Continue
```

EXCEPTION HANDLING

Programming languages use exception handling to detect the common errors and find a way to run the program instead of getting crashed. Writing valid exceptions is an essential skill for all the programmers involved in software development. Exception handling also acts as a great tool to detect both familiar and strange bugs during the testing and maintenance stage of the project.

To handle errors, you need to be aware of try and except statements that can help you to provide a reason for the encountered error to the user.

Example:

Visit your Facebook page and try to insert an image with a size of more than 64MB as your profile picture. After a few minutes of loading, the website will pop you with an error that says, “Image size is too large”.

Here, the Facebook programmers have created an exception handling statement that displays a possible error to the user when image size is exceeded.

Exception handling is also an excellent tool for programmers to learn how a program responds to different instances. Many web and mobile application libraries in Python provide prebuilt exceptions to help programmers quickly create bug-free applications.

Divide-by-Zero Error

When we divide a number by zero in mathematics, it cannot be done and results in an undefined value. Even in programming, it is impossible to define when a number is divided by zero, and therefore all the core programming languages provide divide-by-zero error for their users.

We can use this error type to understand exception handling with try and except statements.

Program code:

```
Def division(value) :
Return 64 / value
Print( value(4) )
Print( value (0) )
Print ( value(16) )
```

Output:

```
16
ZeroDivisionError : Division by zero
```

In the above example, when we try to call a function with an argument of 0, the program ends with a ZeroDivisionError. So, we try to show this error and proceed further with the program using try and except statements.

What Are Try and Except Statements?

Python programmers need to write the potential error block that may occur while executing the program in the 'try' block and should provide a logical solution about what the interpreter should do when it catches an exception in the 'except' block.

Program code:

```
Def division(value) :
Try :
Return 64 / value
Except ZeroDivisionError:
print ( “ A number cannot be divided by zero
” )
Print( value(4) )
Print( value (0) )
Print ( value(16) )
```

Output:

```
16
Error: A number cannot be divided by zero
4
```

Different Types of Errors

Zero division error is only one of the many system errors that Python provides. Knowing about some of these system errors can be a great learning curve to quickly debug or clear your errors.

1. Value error

Value type error occurs when you have provided correct arguments for a function but with a value not supported by the method.

2. Import error

Import error occurs when you try to import a module that is not present at the moment in your directory. You will receive this error with online packages if they are either deleted or disabled.

3. OS error

This error occurs when there is a problem with the operating system that you are working with. These can also be called system-related errors.

4. Index error

Index error occurs with list data types when the index of a list is way out of the bound that the list actually supports.

5. Key error

Key error occurs when a key in a dictionary data structure is not available to process.

6. TypeError

TypeError occurs when a function or object has been given a value of non-supportive data type.

7. Name error

This error occurs when there is no variable provided in either local or global scope.

SAMPLE EXERCISE

Reverse a given integer number using loops.

program code:

```
value = 453
result = 0
print(" What is the number? ", value)
while value > 0:
reminder = value % 10
result = (result * 10) + reminder
value = value // 10
print("Reversed Number is ", result)
```

Output :

```
What is the number? 453
Reversed number is : 354
```

EXERCISES:

Make sure that you create Python code for all these exercises on your own. Only refer to the internet after trying your best and not being able to crack it.

1. Write a Python program to list out numbers up to 2,000, divisible by 12 and multiples of 5. Use separators while listing out the elements.
2. Write a Python program that can convert pounds into kilograms using both for and while loop.
3. Create a random number generator between a number range (1,000 to 10,000) using Python.
4. Use loops to create at least five rangoli patterns
5. Using continue statement, create a Python program that can complete the Fibonacci sequence
6. Write a Python program using loops that can convert USD to both EUR and GBP.
7. Write a Python program that can verify the password authenticity of your input. Make sure you follow password standards to verify them.

Rules:

1. Maximum characters should be 18 letters.
2. You cannot use symbols other than @, $, %, and &.
3. You should use both upper and lowercase letters.
4. You should use at least one letter.
5. Create a Python program that creates patterns of all alphabets. Make sure that you use both loops and conditionals while working on this code.
6. Create a Python program to verify whether or not a year is a leap year and print out all the years that are leap years in the end.
7. Write a Python program that will provide horoscope results based on the date of birth you have supplied as input.

8

FUNCTIONS AND MODULES

A Python programmer needs to be aware of different programming paradigms that are frequently used. Functional programming is popular with Pythonists due to its versatile nature of doing things simply. Writing functions and calling them whenever you need them in a program is easy to follow and effective in runtime execution time. For beginners, understanding functions and learning to implement them is crucial.

WHAT ARE FUNCTIONS?

In mathematics, functions are a binary relation between two sets and are widely acclaimed as many mathematicians' most impactful discoveries. Computer programmers first implemented functions in Z4, the first digital computer. Alan Turing then revolutionized its impact in the computer industry by calling and returning using functions. Functions can also be called methods or subroutines. Every famous programming language implements the usage of subroutines in its core for its users. Python is one of the modern programming languages that enforce functions more straightforwardly.

What Are Functions in Programming?

Programmers usually create programs to complete tasks. To complete tasks, programmers typically need to create an algorithm that completes it in a step-by-step instruction. However, many times a programmer needs to perform a task multiple times. It is not considered a practical solution as it decreases productivity to write and link the same code repeatedly. Hence, functions are created to reuse code to perform different computational and logical tasks.

Functions can also be either user-defined or system functions. Python provides huge libraries of built-in functions that can complete many tasks without rewriting the code again. All these built-in functions of Python are created to be on the point and non-repetitive for programmers.

Example:

Let’s suppose that you downloaded a mobile application that can change filters for your photos. This kind of application uses pre-written functions to manipulate your photos' hue and saturation, resulting in applying a filter to the provided image. Here, the application's programmers created a function to apply these filters to any photo provided instead of doing it all over again for every image.

Almost every interface or component in real-world applications uses functions to create use cases that serve a purpose for the software user. A Python programmer should remember that the prime focus of writing a function is to encapsulate meaningful code to achieve a task.

How Functions Work

The philosophy of functions is simple in all programming languages. First, a programmer writes a block of code that performs a task. Programmers should include this block of code with all the logical and computational logic that a program needs to commit to complete the task.

Now, this block of code can be called with the function's name whenever needed instead of writing the code all over again. Function calling is sensitive and can end up in errors if not done correctly. Functions also contain parameters to call functions with specific input values for more robust results.

How to Define Your Own Functions

Python uses the “def” keyword to create a custom function in the program.

We will now create a small function known as welcome() to explain how a function is defined and called.

Program code:
```
def welcome() :
“This displays a welcome message for the
user”
Print ( “Hi! We give you a warm welcome to
our software” )
welcome()
```
Output:
```
Hi! We give you a warm welcome to our
software
```

Explanation:

1. In line 1, def is used to define a function in the Python programming language. Without writing a function definition, the interpreter will not detect it as a function. A function definition includes def and is followed by the name of the function. Welcome() is the name of the function in the above example. All

the parameters of the functions will usually be placed between parentheses. A colon is placed at the end of the function definition

2. All the lines that follow the function definition belong to the body of the function. Usually, all the programming logic that performs a task is included in this section. The first line of the function body is typically a docstring. Docstring will be used to describe what a function does. While it is not mandatory to use docstrings, it is considered a good practice by many Python programmers. Docstrings are usually enclosed in triple quotes and can be multiple lines.
3. In the above example, the third line uses a print statement to display content on the screen. While in this example, the function has only a single purpose, in the real world, however, a function body usually has more than ten lines to complete a task.
4. The fourth line represents a function call that precisely says to execute the content in the function. When an interpreter finds a function call, it will head over to the written function and run it. As the above example is a function with no parameters, the interpreter will execute simply by printing the statement provided.

USING PARAMETERS IN THE FUNCTIONS

In the previous example, we have defined a basic function with zero parameters. But in the real world, programmers regularly use parameters to improve the functionality of the application.

For example, we can dynamically display the user name using an input parameter in the function.

Program code:

```
def welcome(name) :
“ This displays a welcome message for the
user along with their name”
Print ( “ Hi, “ + name.title() + “ ! \n We
give you a warm welcome to our software” )
welcome(‘tom’)
welcome(‘grace’)
```

Output:

```
Hi, Tom! we give you a warm welcome to our
software
Hi, Grace! we give you a warm welcome to our
software
```

Explanation:

1. In the function definition, we added a parameter called ‘name’ that will accept any value to print on the screen. As Python is not a statically typed language, there is no need to mention the data type of the parameters.
2. In the print statement, we gave the user ‘name.title()’ to provide a value to the defined parameter. If you do not provide a value for the parameter, the program will not execute and end with an error. The title() string function prints the names in a camel case style for better readability. You can use other string functions such as lower() or upper() to achieve different results.
3. In the final lines, we have called the functions with two different parameter arguments. Each argument will display a different output. Here, ‘tom’ and ‘grace’ are the arguments that we have given to a parameter in the function.

The sole difference between a parameter and an argument is that the parameter is essential for the function to work. In contrast, an argument is just a value that programmers can change.

PASSING ARGUMENTS

Functions in real-world applications usually have multiple parameters, and hence you need to know different ways to pass your arguments to the function parameters. While there are several ways to pass arguments, positional and keyword arguments are the most popular.

1. **Positional arguments**

When you use a positional way of passing arguments, you need to match the number of arguments. In positional arguments, the order of passing arguments also becomes essential.

Program code:

```
def sports(country, number) :
'” This describes how many times a country
has won FIFA world cup ‘“
Print ( country + “ has won FIFA “ + number +
“ times”)
Sports(‘Brazil’ , 5)
Sports(‘ France’ ,4)
```

Output:

```
Brazil has won FIFA 5 times
France has won FIFA 4 times
```

In the above example, the arguments are provided at the end of the program using the parameter.

In positional arguments, the change in order can become messy or sometimes result in errors, as shown below.

Program code:
```
def sports(country, number) :
'" This describes how many times a country
has won FIFA world cup '"
Print ( country + " has won FIFA " + number +
" times")
Sports(5, Brazil)
Sports(4,France)
```
Output:
```
5 has won FIFA Brazil times
4 has won FIFA France times
```

The main advantage of positional arguments is that they are easy to call functions using this way, especially when you want to do it multiple times.

KEYWORD ARGUMENTS

Keyword Arguments is another famous way to pass arguments to functions. It usually uses a key : value pair to give arguments to functions. Keyword arguments are mainly used to avoid confusion when passing arguments. They also don't have any order restrictions as the values are directly linked. However, one major defect is that it takes more time and sometimes can become complicated when passing multiple values.

Program code:
```
def sports(country, number) :
'" This describes how many times a country
has won FIFA world cup '"
```

```
Print ( country + “ has won FIFA “ + number +
“ times”)
Sports(country = ‘ Brazil’, number = 5)
Sports(country = ‘ France’ ,number = 4)
```

Output:

```
Brazil has won FIFA 5 times
France has won FIFA 4 times
```

DEFAULT VALUES IN PYTHON

Python also provides an easier way to provide default values for different parameters while creating a function. Using default values while writing parameters is entirely optional. You can either enter it with the parameter or directly enter the argument values in a function call.

Using default values is considered a good practice as it reduces boilerplate code. Boilerplate code is automatic unnecessary code that programmers need to write for an interpreter. Reducing boilerplate code while writing an application can make your code look less complex and increase readability.

Program code:

```
def sports(country, number = 5) :
'” This describes how many times a country
has won FIFA world cup ‘“
Print ( country + “ has won FIFA “ + number +
“ times” )
Sports(‘Brazil’)
Sports (‘Argentina’)
```

Output:

```
Brazil has won FIFA 5 times
Argentina has won FIFA 5 times
```

In the above example, using default values has simplified function calls.

Remember that the Python interpreter will use a default value only when an argument value is not provided during the function call. If an argument value for the parameter is provided, then the supplied argument will be used instead of the default argument value.

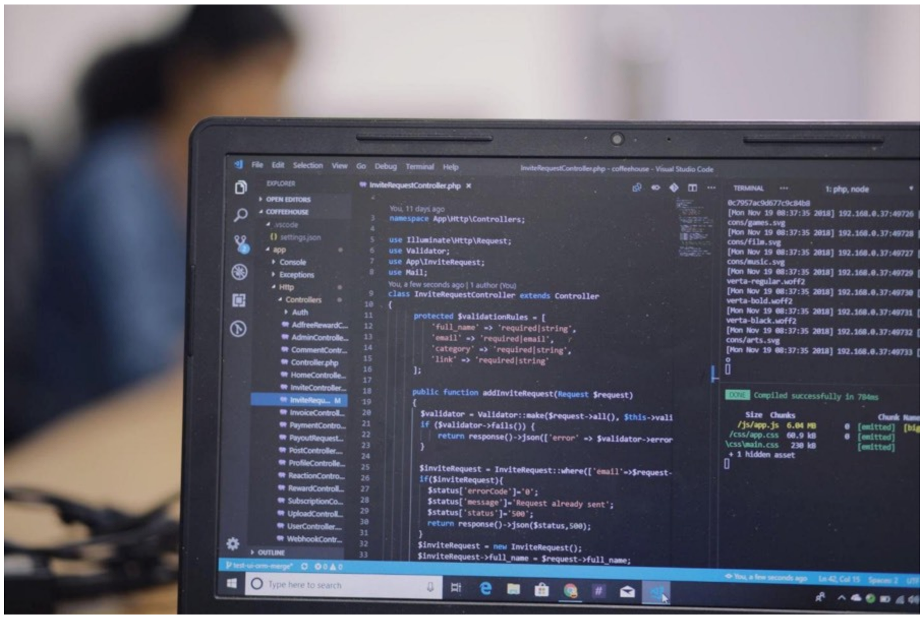
Program code:

```
def sports(country, number = 5) :
'” This describes how many times a country
has won FIFA world cup ‘“
Print( country + “ has won FIFA “ + number +
“ times” )
Sports(‘Brazil’)
Sports (‘Argentina’, 4)
```

Output:

```
Brazil has won FIFA 5 times
Argentina has won FIFA 4 times
```

In the above example, when we called the function with only one parameter, it automatically used the default value. However, when we call the function with two parameters, the default value is replaced by the new argument value.



UNDERSTANDING SCOPE IN PYTHON

Usually, the parameters and variables declared within a function can only be called or used within that function. With respect to function, these variables will be called local scope variables, whereas those declared outside the function will be called global scope variables.

Note: A function is either a local or a global variable; it cannot be both.

Why is a Scope Essential?

The scope is essential because it helps to store all variables given to it while executing a program. For example, if a program is destroyed, the interpreter will forget all the previously given variables, and a new set of variables is required when the program restarts.

Whenever you call a function in the program, a local scope will be created and used until the function returns a value. When the function returns a value, the interpreter will forget all the variables given.

The main reason why programmers prefer to use the scope in their applications is that they can help programmers find the code lines that create a buggy situation in the program. If you use a global scope for all the variables, you may have to recheck all the variables from different functions. Using local and global variables will increase your productivity and can help you maintain big projects efficiently.

Understanding Local and Global Scope in Depth

1. You cannot use local variables in a global scope.

Program code:

```
def vegetable() :
Brinjal = 32
Vegetable()
Print(eggs)
```

Output:

```
Traceback error
```

The example mentioned above will generate a traceback error because you have called a print function with a local variable that doesn't support global scope. Just remember to use only global variables in the global scope.

2. All the local functions, however, can use global variables whenever needed.

Program code:

```
def vegetable () :
Print(brinjal)
Brinjal = 32
Vegetable()
Print(brinjal)
```

Output:

```
32
```

```
32
```

The example mentioned above will print the variables both times as the local functions can use variables with global scope.

3. Local variables of one function cannot be used by any other functions.

Program code:

```
def vegetable() :
Brinjal = 32
Fruits()
Print(brinjal)
def fruit() :
Apple = 21
Brinjal = 42
Vegetable()
```

Output:

```
32
```

The above example is complex and needs to be carefully understood for writing better Python programs.

1. When the function vegetable() is called, a local scope variable of brinjal with a value of 32 is created.
2. Immediately, another function, fruit(), is called, and it creates two local scope variables, 'Apple' and 'Brinjal', with values 21 and 42. Remember that multiple scope values can co-exist in a Python program.
3. Now, the fruit() function is returned, and hence all the local scope variables that are stored will be destroyed

4. In the following line, when a print statement is used, the interpreter prints 32 instead of 42.

Both local and global variables can have the same name. There won't be a conflict while writing programs. For example, a local and global variable named 'sample' can be created without generating any errors. While it is acceptable to use the same names for variables with a local and global case, we recommend you not follow it as it may lead to confusion.

UNDERSTANDING THE 'GLOBAL' STATEMENT

Global statement is usually used to change the global scope variable value from within a function.

Program code:
```
def vegetable() :
Global brinjal
brinjal = ' tasty'
Brinjal = ' global '
Vegetable()
Print(brinjal)
```
Output:
```
Tasty
```

1. As we have mentioned in the global statement for brinjal, whenever a function is called, a local variable will not be created, but instead, a variable with global scope will be created.
2. Hence, when the interpreter executes the print statement, the value of 'tasty' will be printed as it is a variable with global scope.

The usage of scopes in Python programs can be compared to a “black box” where you only need to know the arguments and parameters you are giving. All modern programming languages and libraries will not burden you by asking you to look at all the present global scope variables.

From a beginner’s perspective, all you have to do is understand what the program does and insert the values to that function.

HOW TO IMPORT MODULES

Modules are formed by combining a group of classes. Every program needs modules to complete tasks efficiently. To use modules, you need to import them using the ‘import’ statement. Importing modules is quite similar to '#include' in C/C++ languages. Importing a module usually provides permission to use the methods present in those classes in other programs.

Syntax:

```
import (Module name)
```

Example:

```
import math
```

The above statement imports Python standard library math into the program.

When an import module statement is used, it searches the _imnport()_ function that is used while creating modules. Any operation can be performed easily using this scenario.

Example:

```
import math
Print(math.pi)
```

The above command will first import all the functions present in the math library and then provide you with the literal value of ‘pi’. If not imported, the above print statement will give a not defined error. Also, note down that pi here is a variable in the math module.

If you try to import a module and the Python interpreter cannot detect it, the interpreter will show an import error on the computer screen.

HOW TO CREATE MODULES

The module is a Python component that can consist of different statements and functions in an organized way. For example, suppose you want to create a mobile application. In that case, you can make all the functions related to your user interface (UI) in one module and functions associated with networking in another module.

The advantage of modules is you can easily export them and use them with other applications. Many programmers use modules to create software in team management.

How do you create your own Python module?

1. Create a Python file with a .py extension. For example, we have to create a Python file named sample.py
2. You can now create functions in this file.

Here, I will create a simple function to add two numbers.

File - sample.py
```
Def addition(x,y) :
''' This is a definition that is created to
add any two numbers '''
Z = x + y
Return Z
```

3. Now, you have to create a Python file that consists of a function that adds two numbers. You have also created a module. You can call this with the file name.

Program code:
```
>>> import sample
```

4. As the functions in the module are imported, you can now call the addition function using a dot operator.

Program code:
```
>>> sample.addition(10,20)
```
Output:
```
30
```

In the above example, the Python interpreter uses the addition function in the sample module to provide the result.

Python provides several inbuilt functions and modules to help programmers create code effectively. For example, math is a standard Python library that offers different functions that can perform mathematical operations. Learning about some popular built-in functions and modules can be a great way to understand what you can achieve with Python.

1. print()

Print() is probably the most popular and used built-in function in Python. It prints out whatever is present in between quotes onto the computer screen.

Program code:
```
Print ( “ Hello world”)
```

2. abs()

abs() is a Python built-in function that returns the absolute value for any integer data type. If the provided input is a complex number, then it returns magnitude. In simple terms, it will print the positive value of it if a negative value is provided.

Program code:
```
C = -67
Print ( abs(c))
```
Output:
```
67
```

3. round()

round() is an inbuilt Python function that provides the closest integer number for floating-point numbers.

Program code:
```
C = 12.3
D = 12.6
Print(round(c))
Print(round(d))
```
Output:
```
12
13
```

4. max()

max() is a built-in Python function that will output the maximum number from the provided input values. max() function can be performed using any data type, including lists.

Program code:
```
A = 4
B = 8
C = 87
Result = max( a,b,c)
Print(result)
```
Output:
```
87
```

5. sorted()

sorted() is a built-in function that programmers can use to sort out elements in ascending or descending order. Usually, the sorted built-in function is applied on lists.

Program code:
```
A = (3,34,43,2,12,56,67)
```

```
B = sorted (a)
Print(b)
```
Output:
```
(2,3,12,34,43,56,67)
```

6. sum()

sum() is a particular built-in tuple function that adds all the elements and provides a result to the user.

Program code:

```
A = ( 6,8,45,34,23)
Y = sum(a)
Print(y)
```

Output:

```
116
```

7. len()

Length is an inbuilt python function that will return the length of the object. These are mainly used with lists and tulles.

Program code:

```
A = ( 2,4,5)
B = len(A)
Print(B)
```

Output:

```
3
```

8. type()

Type is an inbuilt Python function that provides information about the arguments and the data type they hold.

Program code:

```
A = 3.2323
Print(type(a))
```

Output:

```
<class ‘float’>
```

STRING METHODS

1. strip()

strip() is a built-in Python string function that returns a result after deleting the arguments provided from the string. Usually, it removes both leading and trailing characters.

Program code:
```
A = “ Hello world”
Print(a.strip(‘wo’)
```
Output:
```
Hell rld
```

2. replace()

replace() is a built-in Python string function that programmers can use to replace a part of a string with another. You can also mention how many times you want to replace it as an optional parameter.

Program code:
```
Sample = “ This is a great way to be great”
Print(sample.replace(“great”, “ bad”)
```
Output:
```
This is a bad way to be bad
```

3. split()

You can use this built-in Python function to split a string according to the separator provided. If no character is provided, the Python interpreter will automatically use white space in its place.

Program code:
```
sample = “ Hello world”
Print( sample.split(‘r’)
```
Output:
```
‘ Hello wo’ , ‘rld’
```

In the above example, the string is split at ‘r’ and is placed in a list as output.

4. join()

The join() method is a unique string function that can help you add a separator to the list or tuple elements. You can use different separators provided by Python or even use custom separators from different Python modules. You can only join string elements in a list. You can’t join integers or floating-point numbers in a list. You can also use an empty string instead of a separator.

Program code:
```
Sample = [‘16’ , ‘32’ , ‘48’ , ‘64]
Result = “ -“
Result = result.join(sample)
```
Output:
```
16-32-48-64
```

SAMPLE EXERCISE

Q. Write a Python function that accepts a string and calculates the number of uppercase letters and lowercase letters.

program code:
```
def sample(s):
d={"Upper":0, "Lower":0}
for c in s:
if c.isupper():
d[“Upper"]+=1
elif c.islower():
d["LOWER_CASE"]+=1
else:
pass
```

```
string_test('This is great')
```

Output:

```
Upper : 1 , Lower : 10
```

SAMPLE EXERCISE

Q. Write a Python class to implement pow(x, n).

program code:

```
class example:
def pow(self, first, second):
if first==0 or first==1 or second==1:
return first
if first==-1:
if second%2 ==0:
return 1
else:
return -1
if second==0:
return 1
if second<0:
return 1/self.pow(first,-second)
res = self.pow(first,second/2)
if second%2 ==0:
return
res*res
return res*res*first
print(py_solution().pow(2, -3));
```

Output:

```
-8
```

EXERCISE:

1. Create a Python program to randomly generate ten numbers and that automatically finds the maximum value in those ten numbers. Use the max() method to solve this program.
2. Create a list and reverse all the elements in it and add them consequently.
3. Write a Python program to input ten strings and reverse each one of them.
4. Write a recursive function to find the factorial of 100
5. Create a 3-page essay using string manipulation techniques. Represent all of them just like how you represent them on paper. Use as many methods as you can.
6. Write a Python program that provides rows related to Pascal’s triangle.
7. Create a Python program that automatically extracts an article from Wikipedia according to the input provided.
8. Create a Python program that can create a color scheme for all the RGB colors.

9

FUNDAMENTALS OF CLASSES

Object-oriented programming is a programming paradigm style that changed how software institutions work. Object-oriented programming provides several practical approaches to writing software when working with teams. Python as a programming language supports both functional and object-oriented programming styles. As a Python programmer, you must understand and distinguish between classes and objects to write a program that represents real-world things and their instances.

WHAT ARE CLASSES AND OBJECTS?

Classes represent real-world things in a program code. They act as a blueprint so that programmers can easily interact with real-world things for different situations. This prototype creates objects, and these object characteristics are usually written whenever you make a class.

Classes help programmers to bundle various functionalities into a single entity. Whenever you create an object from a class, it will have all the behaviors you provided when you made a class. In addition, programmers can give every object unique traits to interact with them more efficiently. Object-oriented programming has an incredible relatability with real-world application use cases, and hence it is currently one of the most popular programming paradigms in the industry.

Classes help programmers reduce the boilerplate code and define the same instances again and again. Along with making things simpler, object-oriented programming also allows programmers to write complex code with a simple instantiation and many instances.

Example:

Let us suppose that you are creating an application for cat breeds using Python. You are collecting information about more than 100 cat breeds and will try to detect them by clicking a picture. To make this happen, without object-oriented programming, you need to input all the information about breeds manually or by using an API. It is still challenging to make this data extracted from an API interact with your application using modular or functional programming even with these complex procedures.

Using object-oriented programming, you will reduce the code repetition by creating a “Cat” class consisting of several instances and methods that will quickly display different cat details using the objects created.

For example, the “age” method in the “cat” class can provide the maximum age of the cat for a particular breed after extracting the data from an API and printing them according to the method. To print the exact same details for a different breed, you need to create another object.

Apart from simplicity, object-oriented programming is a lifesaver for programmers working in teams. Because of classes, they can divide the work in a more organized way. As Python provides different ways to interact with variables inside and outside a class, you can also restrict team members from accessing certain data. Object-oriented programming in Python comes with great customization options that can help you improve your implementation of program logic.

How to Create and Use a Class

Python provides a simple syntax rule to create classes in Python.

```
Class ClassName :
```
Example:
```
Class Cat:
```

Here, “Cat” is the class name. Remember that you cannot use reserved keywords for class names. Let us now create a simple example about cats using Python classes.

Example:
```
Class Cat():
“ This is used to model a Cat.”
Def __init__(self, breed, age) :
““ “ Initialize breed and age attributes “””
self.breed = breed
self.age = age
Def meow(self) :
“”” This describes about a cat meowing “””
print(self.breed.title() + “ cats meow
loudly”)
Def purr(self)
““ “ This describes about a cat purring “””
print( self.breed.title() + “ cats purrs
loudly”)
```

Explanation:

First, we have created a class with the name Cat. There are no attributes or parameters in the parentheses of the class because this is a new class where we are starting everything from scratch. Advanced classes may have many parameters and attributes to solve complex problems that are required for real-world applications.

Immediately after the class name, a docstring called ‘ This is used to model a cat’ describes the necessity for this class. It would be best if you practice writing docstrings more often to help other programmers understand the details about your class.

In the next step, we created an _init_ () method and defined three arguments. This is a unique function that Python runs automatically when an object instance is created from a class. _init_ function is a mandatory Python function; without it, the interpreter will reject the object initiation.

Similarly, we have created two other functions, ‘ meow’ and ‘ purr’, with arguments for each. In this example, these two functions just print a statement. In real-world scenarios, methods will perform higher-level functionalities.

You should have observed the ‘ self’ argument in all the three methods in the above class. self is an automatic function argument that needs to be entered for every method in a class.

In a class, all the variables can be called using a dot operator (.). For example, ‘ self.age’ is an example for calling a variable using a dot operator.

EXERCISES:

1. Write a Python program that will import all the essential classes in Pandas machine learning library.

2. Use a built-in module in Python to list out all the built-in functions that Python supports. Create a Python program to list out all these functions in a table format.
3. Using object-oriented programming, create an OOP model for a library management system. Introduce all the modules that can be used and list out all the arguments that need to be given.
4. Write a Python class of shapes to calculate the area of any figure.
5. Write a class and create both global and instance class variables.
6. Use Python classes to convert Roman numerical standard to the decimal system

WHAT ARE CLASS AND INSTANCE VARIABLES?

When using objects to amplify your program features, you need to use two types of variables: instance and class variables.

Class Variables

Variables are first used to store information with a significant identifier. A class variable is specifically created to ensure that you provide a programming component for a consistent time. When you create a class variable, they can be used anywhere in the program without worrying about unsupported or import errors. All these class variables are usually entered during class construction.

The problem with class variables is that programmers cannot initialize from an instance variable.

Syntax:
```
Class example
```

```
   Game = “ Football”
```

Here Game is a class variable with a value of “Football”.

Class variables can be added and replaced easily, making it an easy option for dynamic Python programs and software programmers.

Instance Variables

Instance variables, on the other hand, are variables that are created explicitly for this purpose. Programmers cannot provide them with variables that are different. Many instance variables are defined within the class itself, and hence it becomes difficult for beginners to understand how they can be replaced or how dynamic allocation can be provided. While there are many ways to create memory allocation for the programs, you cannot use instance variables for universal programs.

You can efficiently work with both class and instance variables to create evergreen software.

WHAT ARE CLASS AND STATIC METHODS?

Functions are different types. They provide value and information to the reader. They can be used to do a simple thing again and again. Programmers can use both class methods and static methods according to Python. Python also provides details about built-in functions.

Class Method

A class method is a built-in method that is evaluated when every function is created. A class method is like a constructor for Python programs and can receive implicit first arguments to make instances repeat easily.

When you use a class method, you cannot find what the object of the class is. A class method is implicit and is also said to receive instant information that is otherwise not controlled and provided by any object in the class.

1. Class methods should be bound to various instance variables that are already present.
2. You should also be able to use a class parameter that can provide multiple information about classes and instances that are present right now in the program cycle.
3. All the variables can be modified and can be called as instances for the class.

Static Method

A static method is a specific programming analysis that python founders provide to help python programmers understand the importance of static arguments common in other high-level languages such as Swift and C.

Many objects present in the class are augmented and cannot be ignored by the instance classes that are present. Not everyone can use classes to decide what their modifying function should be.

To understand the differences between static and instance classes, you need to know both static and instance method constructors. All these constructors have parameters that cannot change because they correlate with functions that are not easy to create.

Many do not try to understand the essence of Parameters as they are created to efficiently extract information from methods. Also, as a Python programmer, you should find different ways to make literals understand why it is essential to use code that can both be changed and extracted using data such as class methods. As a programmer, you also need to be aware of statically supported methods that can interact with standard modules of other Python software.

PYTHON INHERITANCE

Object-oriented programming provides a solution for not repeating code that another programmer has already written. When you write a class, you create different attributes and methods using those attributes. Let us suppose that you need to use that class along with a few other new methods. In this situation, you can use inheritance.

Inheritance is a classic object-oriented programming concept that helps programmers inherit one class's properties to another class. The original class from which a new class is inherited is usually known as a parent class, and the inherited class from a parent class is known as a child class.

Child class, along with inheriting all the attributes and methods present in parent class, can also define new methods for extending the capability of the parent class.

Syntax:

```
Class parent class :
Body of parent class
class child class ( parent class) :
Body of child class
```

For example, let us suppose that a car is a parent class and BMW is a child class, then the inheritance syntax will look as shown below.

Inheritance syntax:
```
Class car
Print ( “ This is a car class”)
Class BMW(car) :
Print ( “ This is a BMW class”)
```

The BMW class now uses all the functions present in the Car class without rewriting them again.

METHOD OVERRIDING IN PYTHON

When you inherit a class, all the functions will be used by a derived class. However, if there are two methods with the same name in both parent and child classes, the interpreter will prefer the child class instead of the parent class.

Inheritance From Multiple Classes

While creating complex applications, programmers will sometimes find ways to create classes that have inherited methods from different classes. In programming terms, this phenomenon is known as multiple inheritance.

Syntax:

```
Class parent1 :
Body of parent 1
Class parent2 :
Body of parent 2
Class child( parent1, parent2) :
Body of child
```

A Python programmer will use super() during these instances to call methods inherited from different parents.

Example:

We will use a mathematical program to explain how inheritance works in Python.

```
Class shapes :
Def __init__( self, sidesprovided) :
Self.first = sidesprovided
Self.sides = [ 0 for result in
range(sidesprovided)]
Def firstside(self):
Self.sides = [ float(input( “ What side is
this?"+str(result+1)+" : "))
for result in range(self.first)]
def secondside(self):
for sample in range(self.result):
print( "Side",result+1,"is",self.sides
results)
```

This class has different attributes and functions that can provide important information whenever there is a change in the sides or when different sides are mentioned.

You can use the methods firstside() and secondside() to display various detailed information about lengths using the shapes class.

As we have a basic overview of important attributes required for a shape class, we are now ready to start a child class called ‘square’.

Program code:

```
Class square(shapes) :
def __init__(self):
Shapes.__init__(self,4)
```

4 because the square has 4 sides. We can use 3 sides if it is a triangle and 5 if it is a Polygon

```
def docalc(self):
First, second,third = self.sides
Result = (first + second + third) / 2
area = (result *(result-first)*(result -
second)*(result-3)) ** 2
print( “The area of the square is %0.8f'
%docalc”)
```

In the above child class, we have mentioned the docalc() function to provide a way to calculate the area of the square quickly. The program has also used various functions from the ‘shape’ class as it has inherited from it.

You can also see instances of method overriding in this example.

How Do We Import Classes?

When writing complex programs, you may sometimes want to import a class from another file to your current class that you are working in. This importing of classes made modules exist in Python.

SPECIAL METHODS IN PYTHON (_REPR_ AND _STR_)

These are special Python functions that are provided by the Python standard library. These special functions are usually helpful for debugging purposes to log the information about different object instances that will start because of a class or function call.

1. Python __str__()

This method is usually used to return what an object represents in a string format. Usually, whenever an object is manipulated using a print() or a string() function, the interpreter will call __str__().

2. Python __repr__()

Using this function, one can try to construct the object again. This will also provide object information in a string format whenever an object is manipulated using any string function. The object information should be in expression format for a better success rate of constructing the object again.

Remember that to call these functions, you need first to use str() and rep() functions. Without them, the Python interpreter will not give results directly.

Many programmers get confused with these two functions because both of them output data in string formats. However, the difference is that the object information due to __str__() function is always in readable format for end-users. In contrast, the object information due to __repr__() function is not in readable format but can be used to construct the object again.

Program code:

```
Sample = date. Determine()
Sample.__str__
()
Sample.__repr__
()
```

Output:

```
'2021-08-01 22:32:22.27237’
'Datetime.datetime(2021,08,22,32,22)
```

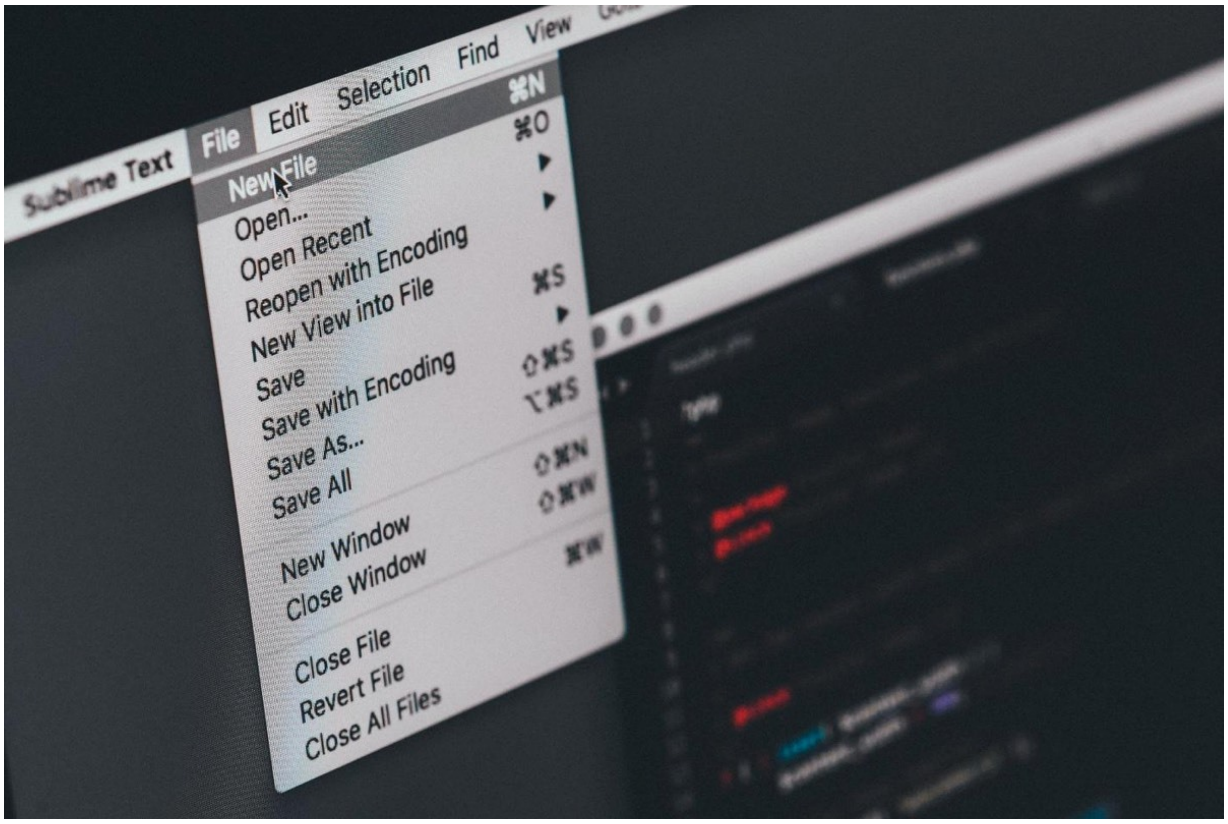
From the above example, we can understand that the __str__ function provides a readable format, whereas __repr__ function is an object code that programmers and testers can use to reconstruct the object. Reconstitution of objects can help programmers quickly debug the code and store those logging instances effectively.

If you are a unit tester, then using both of these functions is extremely important to maintain the program.

11

FILES IN PYTHON

Variables have limited time in a program life cycle. If a variable is ended, then the interpreter will destroy all the data stored in a variable. If the data you are dealing with is sensitive and vital, you don’t want to lose it. To counter this problem of storing and saving data in Python, files have been implemented. Python files can help programmers interact with files that consist of Terabytes of data and modify them according to their convenience. To become an efficient programmer, understanding file operations that are present in Python is essential.



UNDERSTANDING FILES AND FILE PATHS

Any file in any operating system consists of two essential properties. One is the file name which represents its identity and helps users find the content present in it. The other is the file path; this allows users to know where the file is located in an operating system.

For example, if sample.pdf is a file, then ‘C:/users/python/sample.pdf’ is the path for the file. Here, pdf is the extension of the file. A file management system usually consists of folders and directories to effectively organize all the files in the system.

For example, in ‘C:/users/python/sample.pdf’, C is the system's root directory, and both ‘users’ and ‘python’ can be called subfolders in the root directory. Whenever you need to modify or add content to a file using Python, you must explicitly mention the file's path. Otherwise, the command provided will not work and will return you an error.

Note: Remember that the Windows operating system uses backslashes (\) to separate the folder names, whereas both Linux and OS X systems use forward slashes (/) to separate the folders. You can use the os.path.join function to manually enter the path of the file without using back or forward slashes.

Example:

```
Os.path.join(‘C’, ‘ users’ , ‘ python’)
'C\users\python’
```

Current Working Directory

When working with Python files, you often need to interact with the files in your current directory. To help programmers work efficiently, Python provides an easy way to access all the files present in the current directory. All you have to do is use the os.getcwd() function to let the Python interpreter detect your current directory and list out its path.

Example:

```
Os.getcwd()
'C:\\ windows \ program files \ python’
```

Advanced programmers use both absolute and relative paths to access files and manipulate them using Python. However, as a beginner, it is better to stick to the absolute path as it is more straightforward.

Creating New Folders

Whenever you create a complex Python software, it may need to create its directories and folders to efficiently organize different system files. To help programmers solve this problem, Python provides os.makedirs() function, which can create a new directory for the software to interact with.

Example:

```
>>> import os
> >> os.makedirs( ‘ C :\ python\ software\
tutorials’)
```

The above example first imports all the functions from a module named ‘os’ and will search for a method called os.makedirs which can create a new directory according to the path provided by the user.

As a Python programmer, you also need to verify different functions present in the os.path() module to be aware of Python's varying file manipulation methods.

How to Open, Read, and Write Files

Once you know the essential functions, you can directly manipulate files by opening, reading, and writing content to them. You can use both .txt and .py extension files to be run with the Python interpreter. Remember that to interact with particular files such as .pdf or .docx, you need to use third-party Python libraries. These specific types of files are known as binary files according to Python.

First of all, create a file named sample.txt in a new folder. Let us suppose that a file named sample.txt is created in the “ C:/ users/ python/ files/ sample.txt” path. We will now use open(), read() and write() functions to interact with this particular file.

For this example, assume that the data present in sample.txt is as shown below.

Sample.txt:
```
' This is a sample file with a simple text.’
```

Opening Files With the open() Function

To open a file using Python, all you have to do is call the function using the absolute or relative path of the file.

Program code:

```
>>> result = open( ‘. C:/ users/ python/
files/ sample.txt’)
>>> # This will open the file in the Python
terminal
```

Once the open() function is used, the interpreter will open the file in Python read mode. Users can verify and read the information present in a binary file using the reading mode, but the Python interpreter would be unable to read the file. An interpreter will usually open the file with plain-text support. If you are reading pdf or other high-level files, you should install the supported Python modules to read them.

To talk in technical terms, whenever you open a file in Python, the interpreter will form a new file object, and any changes performed for the file will be performed using this instance. If the file is not saved and an instance ends, the interpreter will destroy all the changes.

Reading Files With the read() Function

Now, as the file object file has already been created when you opened it, all you have to do is call a read() function so that the Python software can read the entire content in the file.

```
>>> sample = result.read()
>>> # This makes Python read the content in
the files
>>> sample
```

Output:

```
' This is a sample file with a simple text.’
```

As you have already provided the path before, the interpreter will use it and read it and will read all the content present in a file to either a variable or list according to what you have mentioned.

You can also use the readlines() method to organize the content in a file to separate lines easily.

Create a separate file named ‘example.txt’ in the current directory. Enter the below mentioned data into it.

example.txt:
```
This is a paragraph
But with different lines
We use this example to determine
How Python detects different lines
And list out to us
>>> sample = open(‘ example.txt’)
> > > sample.readlines()
```
Output:
```
\[ ‘ This is a paragraph \n ‘, ‘ But with
different lines \n’, ‘ We use this example to
determine \n’, ‘ How python detects different
lines \n’, ‘ And list out to us \n’]
```

The above example lists all the lines in a list with a new line character (\n).

Writing Content to Files With the write() Function

Python provides an option for programmers to write new data into files using the write() function. It is pretty similar to a print() function that writes output to the screen. The interpreter will create a new object file whenever you open a file using write() mode. You can change the content or append text at the end of the file in this mode. To open the file in write mode, you need to append ‘w’ as an argument to the open() function.

Once you have completed writing the file, you can end it using the close() method. When you use the close() method, your file will be automatically saved by Python.

Program code:
```
Sample = open(‘ example,txt’, ‘w’)
# File now opens in write mode
Sample.write( ‘ This is about working in the
write mode \n’)
```

The output will provide the number of characters in the file. For example, in the above example it is 40.

You can append text to the already written file, using ‘a’ as an argument.

Program code:
```
Sample = open(‘ example.txt’, ‘a’)
# File now opens in write mode
Sample.write( ‘ We are now appending new
content to the file \n’)
# The above-written sentence will be added to
the file
Sample.close()
```

Now, once written, we need to make sure that the Python has read the file to print it on the screen.

Program code:
```
Sample = open(‘ example.txt’)
Result = read(example.txt)
Print(result)
```
Output:
```
This is about working in the write mode
We are now appending new content to the file
```

In the above example, when we print the final result, the written file content and the appended text are printed on the computer screen.

Python also provides a shelve module to quickly interact with data in the file and use them only when needed. To use the shelve module, you need first to import it and once imported, you can call them.

Copying Files and Folders

Using Python, you can use the built-in library known as the shutil module to copy, move, rename, or delete your files. However, like every other library, you need to import the library first.

```
import shutil
```

A Python programmer can use shutil.copy() function to copy files or folders. This function usually has two parameters that ask you to enter both source and destination paths.

Example:

```
Shutil.copy(‘ C;\\ python.txt’, ‘
C:\\python’)
```

This command will make Python create a new file of the same name and copy all the contents in the file.

Here, `C;\\ python.txt` is the path for the source file, whereas `C:\\python` is the path for the destination.

```
Shutil.copy(‘ C;\\ python.txt’, ‘
C:\\python\python2.txt’)
```

This command will make Python create a new file with the name ‘python2.txt’ and copy all the contents from the python.txt.

Moving and Renaming Files and Folders

A Python programmer can use shutil.move() function to move a file from one path to another. Remember that when you move a file, it will be deleted from the current directory entirely and sent to another location. This function uses two parameters to complete the task.

Program code:

```
Shutil.move( ‘ C:\\ sample.txt’ , ‘ C;\\
python’)
```

This command will now move the sample.txt file to another directory.

You can also move by providing a name if you are doubtful that there is another file with the same name.

Program code:

```
Shutil.move( ‘ C:\\ sample.txt’ , ‘ C;\\
python.txt’)
In the above example, we renamed the file
name from ‘sample.txt’ to ‘python.txt’
```

How to Permanently Delete Files and Folders

To thoroughly delete files or folders from a path, you can use the below mentioned built-in Python functions.

1. os.unlink(path) This function will delete a particular file from the path provided.

2. os.rmdir(path) This function will delete the folder name provided. However, remember that the folder should consist of no files.
3. shutil.rmtree(path) This function will delete a folder along with all files and subfolders present in it.

EXERCISE:

1. Write a Python program so that the computer can read the entire text file.
2. Write a Python program and create content using the number of lines.
3. Write a Python program to find consistent details about a variable or a list.
4. Write a Python program that automatically provides a sum to the user.
5. Write a Python program so that we can understand how frequency works with programmers.

ADVANCED PROGRAMMING

First of all, congratulations for completing the sections of this book that deal with the basic concepts of Python programming. With sufficient knowledge about the basics, you are now all set to get a detailed overview of some of the advanced Python programming concepts that will facilitate all the basics you have learned. Understanding these advanced concepts can help you appreciate the ability of Python from different perspectives.

PYTHON RECURSION

Recursion is a process that defines itself. Programming languages often use the same concept to create a function and make them call themselves repeatedly. These functions are known as recursive functions and are often believed to have less runtime as they don't usually interact with other functions.

Recursive functions make the code look simpler and can help programmers organize a problem better. However, it is not always possible to create recursive functions because they are often hard to write and can also take up a lot of computer memory. Recursive functions are also hard to debug because they repeat themselves and don't interact with other variables or program instances.

Finding factorial using Python recursive function:

```
Def find factorial(number) :
“”” This is a recursive function that is
designed to find factorial of any integer “””
If number == 1 :
Return 1
else :
Return ( number * findfactorial*(number-1))
Value = 4
Print( “ The factorial of “ , value, “is”,
findfactorial(value))
```

Output:

```
The factorial of 4 is 24
```

What happened?

In the above example, find factorial() is a recursive function and calls itself. When we mention the number as a positive integer, it uses the provided programming logic to call the function so that the factorial is determined.

PYTHON LAMBDA FUNCTION

A lambda function is a Python function that is defined anonymously in programs. To represent a lambda function in a program, you need to use the keyword ‘lambda’. Python programmers usually use lambda functions to create anonymous functions that have a minimal purpose in the overall program. Lambda functions are also said to have a remarkable correlation with inbuilt python functions such as filter().

Syntax of lambda function:
```
Lambda arguments : expression
```

Remember that lambda functions are encouraged to have any number of arguments but should consist of only one expression.

Example:
```
Sample = lambda result : result * 2 *
Print(sample(5))
```
Output:
```
10
```

ADVANCED DICTIONARY FUNCTIONS

Dictionaries are advanced data structures in Python that can effectively monitor and note down the data.

Dictionaries still need the usage of advanced handling functions such as max(), min() and sort() to achieve some exceptional results.

Using the max() Function

The above example will first check for all the dictionaries key : value pairs to determine the highest value and print out its key. You can also use the max() function with lambda functions.

Program code:
```
Movies = {American Pie : 7 , Schindlers list
: 10 , Shashswank redemption : 9 , Avatar :
8}
Maximum = max( Movies, key = movies.get)
Print(Maximum)
```
Output:
```
Schindler's list
```

Using the min() Function

The above example will first check for all the dictionaries key: value pairs to determine the lowest value and print out its key. You can also use the min() function with lambda functions.

Program code:
```
Movies = {American Pie : 7 , Schindlers list
: 10 , Shashswank redemption : 9 , Avatar :
8}
Minimum = min( Movies, key = movies.get)
Print(Minimum)
```
Output:
```
American pie
```

Using the sorted() Function

You can sort elements present in a dictionary in Python using the sorted() method. While there are different ways, such as using an items() method or dictionary comprehension, the most popular way is to use it as an Iterable argument.

Program code:

```
Movies = {American Pie : 7 , Schindlers list
: 10 , Shashswank redemption : 9 , Avatar :
8}
Print(sorted(Movies))
```

Output:

```
{American Pie : 7 , Avatar : 8 , Shashwank
redemption : 9 , Schindlers list : 10}
```

THREADING

Threading is a phenomenon that allows your Python program to run two instances concurrently without ending any one of them. Without the concept of threading in Python, real-world applications cannot experience a significant speed.

A thread creates a separate flow of execution in your program. Multiprocessing of threads can sometimes need extra knowledge related to Python. To run faster programs, ensure that you import the ‘multiprocessing’ module before implementing a threading function.

To create a separate thread instance in a program, you need to use .start()

```
instance.start()
```

Usually, when you end a program, all the threads will exit automatically. Daemon threads, however, will not exit and will run in the background. Any software that needs the ability to track should use daemon threads not to close the instance completely. Daemon threads utilize significantly less memory power to make them operate for practical implementation.

Some additional threading functions:

1. .join() You can join two threads using this threading function.
2. .lock() You can unlock or lock an instance using this threading function.

PIP PACKAGE MANAGER

Pip is a standard package manager for Python. It is usually used to install additional packages or dependencies that are not present in the standard library. Due to its popularity among programmers, the Python development team has decided to include pip as a default package manager in Python 3 latest versions.

What can you do with Pip?

1. Install new packages and dependencies
2. Find packages from different repositories or Python Package Index (PyPI)
3. Review and install the requirements that are essential for a new package or broken packages
4. Uninstall packages and dependencies with a click

To check whether or not Pip is installed in your system, use the below command.

Terminal command:

```
$ pip – version
```

If it prints out the version details, then the package manager is installed. If not, you need to install it manually from the official website.

Installing Packages

Installing packages using pip is straightforward and takes significantly less time to complete a transaction.

For example, to install http3, a popular Python library, you can use the below mentioned syntax.

Terminal command:
```
$ pip install http3
```

To check the metadata of a Python package, use the below command.

Terminal command:
```
$ pip show http3
```

Output will display all the metadata information such as name, author, license, and location of the package.

To uninstall packages, you can use the below code format.

Terminal command:
```
$ pip uninstall http3
```

To search packages, you can use the below mentioned code format.

Terminal command:
```
$ pip search oauth
```

This command will check for a package named ‘oauth’ in the Python Package Index. You can also manually provide any third-party software indexes to search in them.

VIRTUAL ENVIRONMENT

Usually, when we create Python projects in our system, we use several libraries, all of which have separate dependencies. Sometimes, however, Python programmers need to work with two independent projects that have different dependencies. Python creates an isolated virtual environment to help programmers not mess up their system or avoid errors during these types of situations.

A Python virtual environment can help programmers if they are working with software of two different versions. Python also provides the facility to create an unlimited number of virtual environments.

How to Install Virtualenv

virtualenv is not a standard Python system library. I recommend that you import the package using pip.

Terminal command:

```
$ pip install virtualenv
```

Once the library is downloaded, you can use virtualenv with the help of one of the terminal commands listed below.

Terminal command:

```
$ virtualenv sample
```

When you enter the above command, the interpreter will form a new directory named ‘sample’. It creates new executables and packages that a brand-new application or instance will need.

A newly created virtual environment can be used only after successfully activating it using the below command.

Terminal command:

```
$ source sample/bin/activate
```

Once you enter this command, a new environment will be activated, and you can work on your project now without worrying about any dependencies.

In the same way, you can deactivate a virtual environment using the below command.

Terminal command:

```
(Sample)$ deactivate
```

USING PILLOW LIBRARY TO EDIT IMAGES

Python has great adaptability with digital processing technology and is used in a lot of its applications. You can use a third-party library known as ‘pillow’ to understand the basic essence of creating media manipulation software in Python.

First you must install pillow by using the below command via pip.

Terminal command:

```
pip install pillow
```

As the pillow library is very large, we will import only specific functions that will do the work for us.

Terminal command:

```
from PIL import Image
```

Once the functions are imported, we now need to open an image to work with. Download an image and move it to the pillows folder. You can specify the path of the image if it is in another directory.

Program code:
```
Sample = Image.open(“example.jpg”)
```

Now, you have an image that is ready to be manipulated by other functions in the library. For now, we will learn how to add text to the image using the pillow library.

To make this happen, you need to import a few other functions into your program.

Program code:
```
from PIL import ImageFont, ImageDraw
```

ImageFont function provides a way to download a font according to our requirements to our directory.

Program code:
```
Result = Imagefont.truetype(‘Helvetica.ttf’,
300)
```

In the above Python command, the Imagefont function takes two arguments. The first argument mentions the font name, and the second argument mentions the size of the font.

Now, with an image and font file in your directory, all you need to do is render the text.

Program code:
```
Title = “ This is edited”
Sample.text((23,43), title , (232,342,212) ,
font = result)
```

The above function uses four arguments to render the image and place the text provided on the screen.

In the above command, the first argument mentions the coordinate position of the text. The second argument mentions the text. The third argument uses RGB mode to define what color the text should use. The final argument provides the font style that needs to be used.

MASTERING PYTHON REGULAR EXPRESSIONS (REGEX)

Regular expressions are a special sequence of characters that programmers can use to search using a pattern. Regular expressions are famous among Python programmers due to their robust approach to finding a pattern among a large chunk of data. Python provides a separate module known as the ‘re’ module to handle all the regular expressions.

How to implement a regular expression using Python

1. Specify a pattern string.
2. Provide a regular expression object to pattern string.
3. Find a string pattern using the regular expression object.

Example:
```
Import re
Sample = “ This is an example” _ regex =
re.compile(sample)
Result = regex.search(“ This is an example”)
```

In the above example, we used the search() function and regular expression to find a pattern in the string.

UNDERSTANDING JSON DATA

The Internet is filled with websites. Websites usually communicate with servers to provide data to the user. Servers save both simple and complex data. As a programmer, you often need to communicate between data and server to create groundbreaking applications. JSON provides an easy way to structure the data.

To perform all JSON operations, an inbuilt Python library 'JSON' is provided for Python programmers. Using this library, you can parse, serialize, and deserialize all the objects related to JSON parameters.

How to read JSON files

To read JSON files in Python, you need first to import using the below command.

Terminal command:
```
import json
```

Once the library is imported, you can create file objects, as shown below.

Program code:
```
json.load(file object)
```

To paste the content present in a JSON file, a Python programmer needs to use a loads() function.

Program code:
```
json.loads(content)
# This reads all json functions
```

Apart from making it easy for communicating with the server, JSON also provides an easy way to write metadata that is often required for python web programs. You can use the json.write() function to add new JSON data to a module or application.

UNDERSTANDING SYS MODULE

Python interpreter is an important component in the Python ecosystem. A Python interpreter carefully analyzes every literal, constant, method, and variable to create a step-by-step execution of the logical program. The sys module is provided in the Python standard library to provide advanced information about Python interpreter. You can use the sys module to know sensitive information about constants, variables, and methods.

To use the sys module in your program, you need first to import it, as shown below.

Program code:

import sys

1. To know information about all the existing modules present in this instance, you can use the below command.

```
>>> Print( sys.Argv)
```

2. To know the default system path of Python in the system, you can use the below command

```
>>> Print( sys.path)
```

3. To display the copyright of the program instance, you can use the below command.

```
>>> Print( sys.copyright)
```

4. To display how many times a variable or instance object has been used in the program, you can use the below command.

```
>>> Print( sys.getrefcount(variable))
```

ITERATORS AND GENERATORS

Any process that uses the same logic while executing more than one time is known as an iteration. For example, a loop in a programming language like Python can be called an iteration. If a for loop goes through six times, it can be said to have iterated six times.

In Python, you can use the iter() function to create an iterator object. You can also use the next() function to find the next Iterable object.

Program code:
```
Sample = iter(‘x’,’y’,’z’)
```
Output:
```
X
Y
Z
```

Generators are also a kind of iterator, but they use the keyword ‘yield’ to implement them. Generators will also return all the objects that are repeated more than once as output.

The main difference between iterators and generators is that iterators need a class to implement, whereas generators need a function to implement them.

JUPYTER NOTEBOOK

Python has high adaptability to data science projects due to many third-party libraries that can help data scientists manipulate data in any way possible.

A notebook is a software tool developed for programmers, which can provide an output related to visualizations, media, and equations all in one place. Notebooks help programmers to manage their data effectively. A data scientist can use a Jupyter notebook to make the data science model more engaging and understandable. Because it is an open-source project, there are many integrations available from the community.

How to Use a Jupyter Notebook

To be aware of Jupyter, you should have sound knowledge about Python and Pandas. To install Jupyter in your machine, you can use the pip package manager.

Program code:

```
pip3 install Jupyter
```

Once installed, you can open it and create your first Jupyter. The software will welcome you with an interface that provides details about your notebooks. Installing Anaconda will also help you to change and modify these notebooks easily.

To access your Jupyter notebook, you need to go to a browser and enter the software's URL address. When the URL is opened, Jupyter will start a Python server, and you will now be ready to interact with it as you wish.

The software will save all the Jupyter notebooks in .ipynb format. Many tools will help you to convert these files into traditional docs or pdf formats automatically.

Understanding Cells in Jupyter Notebook

The body of the notebook in Jupyter will usually consist of cells. The most important, however, is the code cell and markdown cell.

1. Code cell

This contains all the Python code that needs to run applications. This code can be converted using the kernel, and an output will be displayed on the screen itself.

2. Markdown cell

This usually contains information about the code in markdown format. Markdown cell usually will facilitate a code cell. Also, remember that you can never create a markdown cell first.

You can edit or write commands on a Jupyter notebook.

Understanding Markdown

Markdown is a special markup language that is created to make formatting the text easy. Programmers can use it to write both programs and normal text material.

Every markdown element has its own set of rules. For example, # will create an H1 heading, whereas ## will create an H2 heading.

UNIT TESTING

Whenever programmers write code, they need to ensure that the code quality is on par with Python's guidelines. Even if it works, complex and messy code can create bugs down the road and may cause bottleneck situations. Usually, when a programmer completes software, he needs to conduct test case execution. During this phase, programmers should analyze all log tests and reports to be aware of all potential warnings that may occur.

Normally, programmers need to write manual testing code to verify the authenticity and maintainability of their code. However, manual testing is a humongous task and can sometimes even take more time than the development phase. Python provides a unit test framework known as “unittest” to test the code automatically to make programmers' lives easier.

How Unit Tests Take Place

To conduct a unit test of your code, you must first decide how to address your testing procedure. For example, you can test an entire module at once, or you can conduct testing exclusively on a single function. However, it is recommended to start testing with small code and expand further.

What functions does the unit test offer?

1. A Python programmer can automatically test various parts of a software.
2. You can share test code easily with other developers. Python interpreter will also share all the build and runtime errors.
3. You can organize a group of tests and can call them collections. This strategy can help programmers to maintain the software for a long time effectively.
4. Programmers and testers can easily manage testing with independent frameworks.

Unit test modules provide assert() methods to perform various tests. For example, assertcountequal() will check whether or not elements are in order.

To write efficient programs that can work well even in difficult instances, you need to know complex unit testing methods.

GITHUB FOR PYTHON PROGRAMMERS

GitHub is a GIT-based resource allocation. It is a peer-to-peer website that programmers can use to coordinate with projects effectively. GitHub offers both free and professional versions for its users. All your repositories will be public as a basic user, and your code instantly becomes an open-source code. On the other hand, if you choose a professional version, you can work with your team using private repositories protected using high encryption algorithms.

Why is GitHub essential for beginners?

GitHub can be a great way for beginners to read more efficient open-source code created by experienced programmers. There are also many projects that beginners can use to improve their programming and logical reasoning skills. Many Python libraries are hosted in GitHub, and hence exploring it is a mandatory prerequisite for beginners. You can use several third-party web or mobile clients to import git projects to your local system instantly.

Git supports version control and is hence an easy to update your project dependencies. You can either commit or change the code using git code with an instant click.

To create a new repo, you need first to enter your git console using the below command.

console code:
```
$ git config –global root “Project name”
```

Once you enter a git console, enter the below command to create a new directory.

console code:
```
$mkdir. (“Name of the Repository”)
```

You can also find additional information about a git repository using the below command.

console code:
```
$ git status
```

You are now all set to experiment with your Python code and create open-source projects.

DIFFERENT PYTHON LIBRARIES

The enormous number of Python libraries is one of the main reasons for the popularity of Python as a programming language. Libraries make work easy for developers to create real-world applications. Knowing about some of the popular Python libraries that can create efficient Python software is essential for any programmer.

1. Requests

Requests is a Python library that is developed to create easy HTTP requests to a specified URL. As Python is a popular programming language to create websites and develop scrappers, programmers must use a reliable library to create both requests and responses in a simple yet effective way.

Requests also help programmers to interact with the data that APIs provide. While all the data is usually provided in JSON format, the requests library can easily read and update this data.

You can install the requests library from the Python package index using the pip package manager.

Program code:
```
pip install requests
```

2. Scrapy

Web scraping is a popular way to extract data from websites consistently and automatically. Scrappers also sometimes can interact with APIS and act as a crawler. Scrapy is a Python library that provides the ability to scrape the web data for a Python web developer quickly.

Usually, web scrapers use ‘spiders’ to extract the data, and as a Python library, Scrapy extends the ability to scrape using advanced spiders. Scrapy also was inspired by Django; it doesn’t repeat any code already present in the library. All Python developers can use Scrapy shell to provide assumptions about the behavior and performance of a site.

3. SQLAlchemy

The internet is filled with relational databases that save the data and provide different mapping technologies to interact effectively. All websites and applications use SQL databases more due to their data mapper technology.

SQLAlchemy is a Python library developed to act as an object-relational mapper for software developed using Python. SQLachemy also operates an active record pattern to detect data mappers present in a database. Python programmers can also install additional plugins to automate the constant maintainability that comes with relational databases.

4. NLTK

Natural language processing (NLP) is an artificial intelligence technology that implements cognitive ideas in a programmatic sense. NLTK (Natural Language Toolkit) is a set of Python libraries that specialize in NLP technology. They are primarily written in the English language and provide various graphical demonstrations according to the data provided.

At first, programmers developed this Python library to use it as a teaching resource. However, its adaptability to domains apart from NLP such as artificial intelligence and machine learning had made it popular among data scientists. It provides features such as lexical analysis and named entity recognition.

5. Twisted

To write any internet-related software in Python, a programmer should be aware of different networking concepts and should find a way to support them so that software can interact with them. A Python framework called Twisted was developed to give programmers an alternative to having to create networking protocols from scratch every time.

It supports different network protocols such as TCP, UDP, HTTP, and IMAP. It uses an event-driven programming paradigm to create callbacks whenever the framework is used. It provides advanced functions such as foreign loop support, threading and deferreds. Twisted is the primary networking library for many famous websites such as cloud ki, Twitch, and Twilio.

6. NumPy

Python was created to develop desktop and web applications, but not for numerical or scientific computing. However, in 1995 many scientists found that Python’s simplicity can help it effectively correlate with mathematical equations. Once the implementation of arrays has been announced, Python has started to get adopted to various scientific and numerical libraries. NumPy supports the usage of high multidimensional arrays, thus making it one of the important python libraries for performing high-level mathematical functions.

NumPy provides high debugging and binding features for several other libraries, and hence it is often used with other popular scientific and machine learning libraries while writing Python software.

7. SciPy

In the last decade, Python has become the most used programming language in scientific and technical computing. Due to its open-source nature, many computer scientists believe that using Python as the default language while creating scientific and technical applications can make them interact better as a community. SciPy provides modules for different scientific domains such as optimization, interpolation, and image processing.

SciPy also correlates with NumPy while working with high multidimensional arrays that use mathematical functions. SciPy’s impact on digital imaging and signal processing also made it a favorable choice for technical computing.

8. Matplotlib

Matplotlib is used as a companion library for SciPy to plot all the mathematical equations and high-level mathematical functions using charts. While looking at the equations and functions can develop your scientific and technical skills, a chart can make you interact with data more clearly. Python programmers use Matplotlib in their software so that the end-user will understand the purpose of the software more efficiently. Many also use Matplotlib as a reference tool for unit testing different frameworks more easily.

Matplotlib uses GUI toolkits such as Tkinter to represent plots beautifully. Additional functions such as GTK tools, Excel tools, Basemap, and cartopy had made it a critical Python library for programmers involved with technical and scientific computing.

9. Pillow

Pillow is a Python library that programmers can use to manipulate images. Pillow provides different modules that can add additional image-enhancing capabilities to the software. Pillow is an updated library for the already famous Python Image Library (PIL). PIL is a legacy project and has been discontinued from further development. To utilize the functions that PIL comes with, enthusiastic developers have forked down in Pillow's name to work with Python3. Forking down refers to copying an old library into a new library with added functionalities.

Pillow supports various image files such as jpeg, png, tiff, and gif. With a pillow, you can crop, rotate, manipulate, or edit images. Several hundreds of filters are also available for Python programmers with a few clicks.

10. Beautiful Soup

Beautiful Soup is a Python scraping library that programmers can use to customize how to parse both HTML and XML pages. Beautiful Soup creates a detailed parse tree after scraping and extracting content from these web pages. It also uses a markup language to perform its parsing operations. It is also known as a beautiful parser due to its ability to scrape data without being messy while validating and conforming information.

Beautiful Soup also works with all of the latest web frameworks and technologies along with HTML 5 elements. If you are a web developer who needs to create software that interacts with web elements, there is no better Python library than Beautiful Soup.

11. WxPython

WxPython is a Python wrapper to the cross-platform popular GUI library wxWidgets. wxWidgets is written in C++ and provides many tools that can help programmers create beautiful desktop applications. Enthusiastic Python developers have created a wrapper to utilize the universal functionalities provided by wxWidgets to build desktop and mobile applications.

It has different modules for widget development. For example, a button module can help you create buttons of various kinds with a small code.

12. PyQT

PyQT is a Python extensible library that supports the popular cross-platform GUI library Qt. It can be installed in Python using a simple plugin. It provides modules that programmers can use to develop Windows, Linux and macOS applications. PyQT is also portable and can be used instantaneously in any operating system.

It has a substantial set of various high-level GUI widgets, along with an XML and SVG parser. It also provides different modules for regular expressions, Unicode modelling, along with DOM interfaces.

13. Pygame

Python also has high adaptability with mid-range gaming technology. Pygame is a popular Python library that provides various computer graphics and multimedia libraries that programmers can use to write video games. It offers multiple modules to manipulate the gaming experience with the help of sound, keyboard, mouse, and accelerometer.

Python programmers can also use Pygame to create games that work with Android smartphones or tablets. It supports the SDL library that helps programmers to create real-time interactive games with few lines of code.

14. Pyglet

Pyglet is a Python library that programmers can use to create advanced games and multimedia software. Pyglet is highly portable and can help programs to run in all operating systems without any lag. It extends multimedia software capabilities by providing full-screen control, multiple monitors, and different format types.

Pyglet also comes with an additional Avbin plugin that supports various audio and video formats with perfection.

15. Nose2

Nose2 is a Python framework that can automatically detect unit tests and execute them with a mouse click. While there are many testing frameworks, Nose2 provides rich plugins that can make the process swifter. Nose2 also provides better API facilities that can be interlinked with selenium website testing.

Using Nose2, you can also run different unit testing processes simultaneously, making it a possible option for projects with large code. Using Nose2, you can capture log messages, cover test reports, and organize information with different layers.

16. Bokeh

Bokeh is one of the popular Python libraries extensively used by data analysts to create visualisation data with several effects. This Python library is exclusively developed for building visualizations for modern web browsers such as Safari and Opera. Bokeh not only supports high-level interactive graphics but also can be used for creating complex plotting graphs.

Many programmers use Bokeh and streaming datasets to generate high-level graphic visualizations that make them more attractive and intuitively beautiful. Bokeh has a high-level relatability with Javascript technology.

17. Pandas

Pandas is a Python library that is primarily used for data manipulation and can perform various analysis techniques. It helps data scientists to import data sets from different high-level formats such as JSON, SQL, and Excel so that these analysts can perform operations such as merging, data cleaning, and other high-level analysis features.

Pandas is also excellent in terms of aligning data to a handler and performing hierarchical regeneration techniques. Many programmers compare Pandas to C language libraries due to their method implementation.

18. Scikit learn

Machine learning is one of the popular computer technologies that is expanding its horizon to different Python libraries. Sci-kit learn is one of the popular machine learning libraries in Python that programmers can use to create machine learning models. It also supports neural network algorithms in dense situations.

Sci-kit learn has a remarkable correlation with numerical and scientific libraries such as NumPy and Scipy. Random forests, clustering, and K-tree are some of the famous algorithms that Sci-kit learn consists of.

19. Keras

Neural networking is a computer science domain that utilizes cognitive concepts in a programmatic sense. Keras is an open-source Python library that uses Python to create artificial neural networks, making complex real-world applications.

It also supports linking to back-end software such as TensorFlow and Microsoft cognitive toolkit. You need to be aware of different GPU and TPU units to create software using Keras as your primary framework carefully. Keras can also be used as a preliminary framework for deep learning projects.

20. Tensorflow

Tensorflow is a popular machine learning library that is used to create deep neural networks. Tensor flow uses techniques of differentiable programming to create code that can co-relate with data flow structures that exist on the Internet. Tensorflow was first developed by Google and later was adapted to be open source due to its popularity among Google developers.

Leave a 1-Click Review

I would be incredibly thankful if you could take just 60 seconds to write a brief review on Amazon, even if it's just a few sentences!

https://www.amazon.com/review/create-review/?ie=UTF8&channel=glance-detail&asin=B09FP46GLF

CONCLUSION

Congratulations on completing the book. You now have acquired skills that can make you work with mid-level Python projects. To become proficient in Python from now on, you must write quality code consistently. To improve your programming skills, challenge yourself by working on advanced projects.

To be an efficient programmer, you need to be aware of certain habits and tips that experienced programmers use and have referenced as having helped them become better programmers. Any beginner starting with computer programming can use them to improve their efficiency in the subject.

1. Build a strong foundation with fundamentals

While learning syntax basics within a couple of hours and working immediately with projects may seem exciting and challenging, it is not a recommended learning technique. For a programmer, fundamentals are essential. For a Python programmer, it is more critical because Python relies on simplicity. If you want straightforward readability, you need to be aware of all the fundamentals it provides for the end-user.

2. Break problems into smaller ones

Every program solves a problem. Any problem can be solved using different approaches. To write better programs, you need to find the best approach that reduces execution and run time. To find the best approaches, you need to take a large problem and divide it into a set of more minor problems. Approaching solutions using this technique can help programmers create software with fewer bugs and less unit testing. Always use pseudo-code instead of programming code while trying to find approaches or solutions to a problem. Once you find a working solution, you can start implementing it using your favorite programming language, Python.

3. Find your niche

Programming is overwhelming if you can’t find out what niche you love the most. It is not practical to learn a little bit of everything as there are many domains out there in computer science. If you like designing and developing websites more than extracting data and analyzing it, then you should focus more on website development instead of data analysis. Python has the adaptability to any niche, so you can use it anywhere without any problem.

4. Get addicted to errors

Errors are a great way to improve your expertise in a programming language. Whenever you encounter an error, try to Google it using the traceback error given by the interpreter. Googling an error can provide you with information about why the error has occurred and different ways to get rid of the error without messing up the program code already written. Programming is fun if you constantly face hurdles and jump them with a little bit of tinkering.

5. Learn algorithms and be aware of problem-solving techniques

While programmers can solve simple problems with a basic traditional approach or by using open-source code, you need to be aware of different algorithms and problem-solving techniques to solve new complex problems while developing software. Make sure to refer to different algorithmic approaches using advanced Python books and resources. As a programmer, to improve your logical programming skills, we recommend you read a lot of code. Reading a lot of code will not only help you to find out solutions subconsciously but can also help you develop a consistent programming style.

Learning Python is fun and exciting. With the information and knowledge provided in this book, you are now ready to start exploring the world of programming. We hope you develop a lot of software that moves the world forward in a better way. All the best!

REFERENCES

Shruthitv. (2018, August 10). *Python variables* . GeeksforGeeks. https://www.geeksforgeeks.org/python-variables/

Sharma, Rohit. (2019, December 16). *Top 7 data types of Python | Python data types* . UpGrad Blog. https://www.upgrad.com/blog/top-7-data-types-of-python-python-data-types/

Python conditional statements: IF... Else, ELIF, Nested IF & Switch Case . (2019, October 23). Guru99.com. https://www.guru99.com/if-loop-python-conditional-structures.html

Loops in Python . (2017, June 7). GeeksforGeeks. https://www.geeksforgeeks.org/loops-in-python

File handling in Python . (2017, October 10). GeeksforGeeks. https://www.geeksforgeeks.org/file-handling-python/

All images sourced from Unsplash. (2000). https://unsplash.com/