Apply introductory programming techniques

Contents

_Toc61596951
Establishing the application task....4
Clarify task with required personnel....5
The required personnel....5
Responsibilities of the required personnel....5
The principles of good application usability....5
Identify design specifications according to task requirements....6
Design specifications....6
Programming standards....6
The concept of computer programming....7
What is a programming language?....7
What is Python?....8
Advantages of Python....9
Disadvantages of Python....10
How to Install Python on Windows....11
Install Python....11
Step 1: Download....11
Step 2: Installation....12
Writing your First Python Program....12
Step 1: Open PyCharm IDE....13
Step 2: Create a new project....14
Apply Language Syntax and Layout....18
Apply basic language syntax rules....18
Python Syntax Rules....18
Create code using data types, operators and expressions....22
Data Types in Python....22
Setting the Data Type....23
Setting the specific data type....23
Operators....24
Types of Operator....24
The Arithmetic Operators....24
Unary Operators....26
Addition....26
Subtraction....26
Multiplication....27
Division....28
Modulo....29
Power....29
Boolean Operators....30
Relational Operator....30
Python Object Identity Operators....31
Python Bitwise Operators....31
Expressions....32
Apply variables and variable scope....33
What are variables in Python?....33
What is the variable scope in Python?....33
How to Declare and use a Variable....33
Re-declare a Variable....33
Concatenate Variables....34
Local & Global Variables....35
Delete a variable....38

Global Variables 38
Local Variables 39
Global and local variables 40
Nonlocal Variables 41
Use program library functions 43
Clarify code using commenting techniques 50
Writing comments in Python programming language 50
Control structures 51
Apply language syntax in sequence, selection and iteration constructs 52
Sequential 52
Selection 52
Repetition 52
The conditional statements 53
Python if..else statements 54
Python while loop Statements 55
Create expressions in selection and iteration constructs using logical operators 56
Logical operators 56
Logical AND operator 57
Logical or operator 59
Logical not operator 60
Code Using Standard Algorithms 61
Develop algorithms using sequence, selection and iteration constructs 61
Nesting IF constructs 62
Pseudocode—nested constructs 63
Designing algorithms 64
Create and use data structures 64
ARRAYS 64
What is an Array? 64
Access the Elements of an Array 64
The Length of an Array 65
Looping Array Elements 65
Adding Array Elements 65
Removing Array Elements 65
Array Methods 66
Code standard sequential access algorithms used in reading and writing text files 67
Apply string manipulation 68
Strings in Python 68
Basic String Operations 68
String Operators 69
String Methods 69
String Formatting 71
f-strings 72
Test Code 73
What Is Testing? 73
Types of Software Testing 73
Alpha Testing 73
Acceptance Testing 74
Ad-hoc Testing 74
Beta Testing 74
System Integration Testing 74
Unit testing 74
User Acceptance Testing (UAT) 75
White Box Testing 75

Black Box Testing....75
Performance Testing....75
What Is Debugging?....75
Requirements for Debugging....76
Principles of Debugging....76
Use debugging techniques to detect and correct errors....77
Prerequisites....77
Create and conduct simple tests and confirm code meets design specification....78
What is design specification?....78
Document actions carried out and results of tests performed....80
Create a Simple Application and Seek Feedback....82
Design an algorithm in response to basic program specifications....82
Develop application to meet program specification....83
Overview....83
Hangman Script....83
Meeting specifications....86
References....87

Establishing the application task

Before even starting to consider how to program a software application, a person needs to conduct detailed research into what the application is required to do. This initial research needs to answer the following questions:

- What is the application is all about?
- What is required in the application?
- What does the application do in that workplace, i.e. what is its' purpose?
- When is it needed?
- How should it work?
- What should it look like?
- Where will it operate?

...and the questions go on.

Building up a clear picture of that the application must do and how it must work is akin to drawing a road map – the design and development of your application must have a 'map' established before you commence work on the programming.

As a developer, you must set the direction for the work so that you can ensure that the needs and requirements of the end-users are met. Just as a driver might encounter roadblocks or take detours to reach the destination, the developer may also be subjected to change, both internally from their own creative thought or work capacity and from external input (i.e. the end-users, changes to industry standards, etc.). During a development process it is not uncommon for the developer or developing team to conduct a review of the development process to ensure all is on track and the result will be a useful application that meet's the clients' needs.

The Scope Statement is an essential element in the development of a software application. This statement forms the foundation on which the development of the application occurs. It describes, in project terms, the application development and is the document that stakeholders record their agreement to the scope of the work. The Scope Statement also provides detail on what the result from the project will be.

By clearly articulating the scope of the project, people's commitment and responsibilities and the result of the work this reduces the chances of miscommunication.

The Scope Statement should include:

- The client organisation and their needs
- Objectives of the project, which state what will happen in the project to solve the business question, for example, designing and developing a web application
- Justification and benefits of completing the project
- The scope of the development work, i.e. specifications of application, content, specific deliverables, etc.

- Main goals, the strategy and other elements determined by the project's size and nature.

The Scope Statement forms the basis of the contract between application developer, manager and client. The client is generally the only party that has the authority to approve changes to the scope statement.

Clarify task with required personnel

The required personnel

Everyone who is involved (directly or indirectly) in the development project can considered as the 'required personnel'. They may include:

- supervisor
- manager
- vendors
- subject matter experts
- programmers
- developers
- software testers
- other employees within the organisation

Responsibilities of the required personnel

The personnel will review and check the application according to the set requirements, standards and conditions.

They will also provide necessary feedback and helpful suggestions to make the required changes to the application and where change to the scope of the project may occur, these suggestions would go to the client for their approval.

It is always essential that the required personnel are checked and informed before accepting a scope statement. You need to ensure the workers are available and have the necessary knowledge and skills to complete the job, before rushing in and trying to sell what may be 'vaporware'.

The principles of good application usability

The five principles of good application usability describe the development or construction process of the application.

The required personnel will be responsible for reviewing the application according to the following main principles of good application usability:

Availability and accessibility: how user friendly is the application and can it be used effortlessly by the end-user and other stakeholders;

Clarity: how clear or lucid is to understand the design, layout, interface, using navigation panel, etc.;

Learnability: how quickly users can pick up the best and easy ways to use the application;

Credibility: have the application being developed as credible and trustworthy;

Relevancy: how relevant is the application to the user's needs and requirements.

Identify design specifications according to task requirements

Design specifications

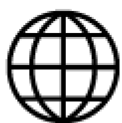
A Design Specification is a structured document that lays out the critical processes and requirements related to the design of an application or product. For example, the design specification typically includes technologies required, development tools, dimensions needed, environmental factors, ergonomic factors, aesthetic factors, maintenance necessary, and many more requirements.

Programming standards

Computer programming standards are programming methods that have been declared acceptable. They are recommended as the approach that should be used to write any programming code or to develop any software. The programming standards require programmers to use a common ground when writing code. This ensures that there is a universal arrangement of code and standardization of component development.

A programming standard gives a uniform appearance to the codes written by different developers. It improves readability, and maintainability of the code and it reduces complexity also. It helps in code reuse and helps to detect errors.

An example of this standard arrangement can be seen at the following site:



PHP Programming standards *(https://developer.wordpress.org/coding-standards/inline-documentation-standards/php/)*

Python – Style Guide PEP 8 *(https://www.python.org/dev/peps/pep-0008/)*

The concept of computer programming

Computer programming is the process of creating a set of instructions that tell a computer how to perform a task. Programming can be done using a variety of computer programming languages, such as JavaScript, Python, and C++.

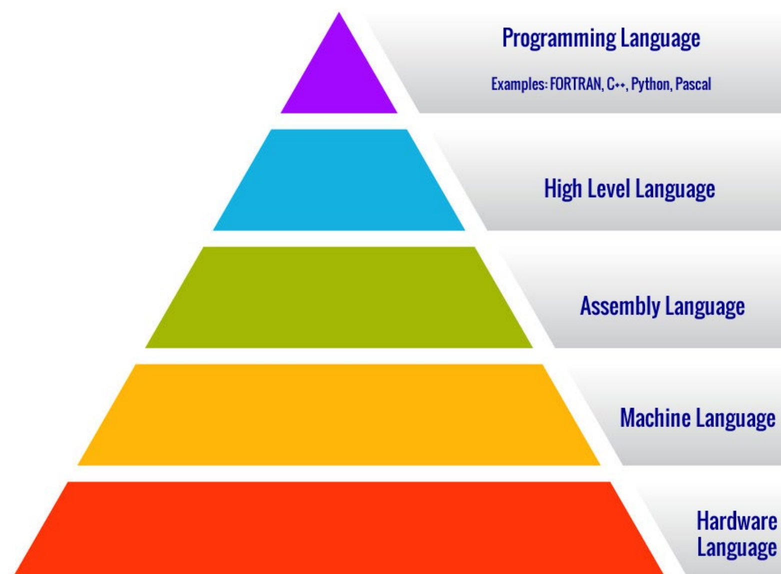
These instructions are known as code and code is written by computer programmers/developers to perform a specific function. Code is the fundamental basis of computing. Whether you are running a social media app on your smartphone or operating with the API of a cloud service, the task is dependent upon the code that has been written in one of many programming languages.

Most historians consider Ada Lovelace (1815 – 1852) as the first programmer in the world. She wrote an algorithm for an Analytical Engine by Charles Babbage. Despite never having completed this machine, Lovelace stated that "*Mr Babbage believes he can, by his engine, form the product of two numbers, each containing twenty figures, in three minutes.*" Although this is slow Babbage and Lovelace were well ahead of their time. It was not until the 1940s that digital, programmable computers appeared.

In 1970, Niklaus Wirth developed the language known as Pascal, which is still used to make desktop applications for Skype; in 1983, Bjarne Stroustrup developed the object-oriented language C++, which today powers Google's Chrome web browser, among others; and in 1991, Guido Van Rossum contributed the extremely useful and efficient Python language, named for the British comedy group Monty Python.

What is a programming language?

To create a software program/application, a programming language is used to compile a set of commands, instructions and other syntax uses. Languages used by programmers to write code are considered "high-level languages," which can be translated into a "low-level language" that is immediately understood by the computer hardware. A programming language with its own set of vocabulary and grammatical rules for instruction perform tasks on a computer or computing machine device.



High-level languages are designed for fast reading and comprehension. This allows programmers to write source code, using logical expressions, words and symbols. For example, reserved words for loop statements are used in most major programming languages such as while, if and else. Symbols as <, >, = =, and! are commonly used as operators. Most high-level languages are similar so as to allow programmers to understand source code written in multiple languages with ease.

High-level languages, for example, include C++, Java, Perl and PHP. Languages such as C++ and Java are called "compiled languages," since the source code must be compiled for running first. Languages such as Perl and PHP are called "interpreted languages," since the source code can be executed without being compiled through an interpreter. Compiled languages are typically used to build software programs, whereas interpreted languages, such as those used to produce content for interactive websites, are used to run scripts.

Assembly and computer languages are included in the low-level languages. A language of assembly contains a list of basic instructions and is much harder to read than a language of high level. A programmer can, in rare cases, decide to code a simple program in an assembly language to ensure that it runs as efficiently as possible. The assembly code can be translated into computer code using an assembler. The machine code, or machine language, contains a series of binary codes which a computer's CPU understands directly. Computer language is not intended for human readability but written for computer machines.

Hundreds of different programming languages have already been developed, and more are created regularly. Most programming languages are written in an imperative style (i.e. as a series of operations to be performed). In contrast, other languages use the declarative style (i.e. you decide the desired outcome, not how to achieve it).

Generally, the definition of a programming language is divided into the two components of syntax (form) and semantics (meaning). Many languages are specified by a standard document (for example, an ISO Standard specifies the programming language for C). In contrast, other languages (such as Perl) have a dominant implementation that is regarded as a reference. Many languages have both common, with a standard specified basic language, and extensions taken from the dominant implementation.

What is Python?

Python is a powerful, general-purpose, object-oriented, high - level programming language with dynamic semantics. Combined with dynamic typing and dynamic binding, its high-level data structures make it very appealing for Rapid Application Development. It is also valued as a scripting or glue language for connecting existing components together. Python is designed for readability and to be close to the English language. It is easy to write syntax that emphasises the readability of the code and this in turn reduces software maintenance costs. Python supports modules and packages which promote modularity of the program and reuse of code. On all major platforms, the

Python interpreter and the comprehensive standard library are available in source or binary form at no charge and can be distributed freely.

Programmers also fall in love with Python because of the flexibility it delivers. The edit-test-debug process is incredibly quick because there is no compilation phase. Python debugging programs are simple: a bug or bad input will never trigger a segmentation fault. Instead, it creates an exception when the interpreter detects a mistake. If the program fails to capture the exception, the interpreter will print a stack trace. A source-level debugger helps you to inspect local and global variables, evaluate arbitrary expressions, set breakpoints, walk through the code one line at a time, etc.

The debugger itself is written in Python. On the other hand, sometimes the easiest way to debug a program is to add a few print statements to the source: the speedy process of edit-test-debug allows this simple approach quite efficient.

Advantages of Python

There are advantages to Phyton as a programming language. The major advantages are summarised below:

Extensive support libraries: Python downloads with a robust library containing code for various purposes such as regular expressions, documentation creation, unit testing, web browsers, threading, database, CGI, email, image manipulation, etc. Therefore we don't need to write the whole code manually for specific functions and purposes.

Integration Feature: Python incorporates Enterprise Application Integration, which enables the creation of Web services by activating COM or COBRA components. The language has powerful control capabilities, as it calls via Python directly through C, C++ or Java. The advantage of Python processing the XML and other markup languages is quite an important feature, as the programming language can run in the same byte code on all modern operating systems.

Extensible: Python, as we have stated before, can be extended to other programming languages. You have the flexibility to write and use some of the Python code using languages such as C++ or C. This feature is quite useful, especially when you are working on projects and their features.

Improved Programmer's Productivity: Python language has comprehensive support libraries and clean object-oriented designs that improve programmer efficiency by two to tenfold, when using languages such as Java, VB, Perl, C, C++ and C #.

Embeddable: Python is also embeddable, as complementary to the extensibility. You can transform your Python code into a different language in your source code, such as C++. This feature is helpful to add scripting capabilities (using other languages) to our code.

Productivity: With its solid process management functionality, unit testing framework and improved control capabilities lead to increasing speed for most applications and application productivity. It provides an excellent way to create multi-protocol network applications which can be scaled.

Internet of Things Opportunities: Because Python forms the foundation for new platforms such as Raspberry Pi, it provides a bright future for the Internet of things (IoT). This is a way of linking the language to the modern world. Python is the right option in IoT systems for data analysis. The language is plain and easy to deploy. The wide user community is willing to assist others and extensive libraries exist. It is the ideal language for data-intensive applications.

Simple and Easy: Training, understanding and programming are relatively simple. When people start to use Python, they find it hard to adapt to other more verbose languages such as Java.

Readable: Using Python is just like reading English because it's not such a verbose language. That is one of the reasons why studying, understanding and coding in Python are so simple. It is strong in code readability.

Object-Oriented: Python language follows paradigms for both the procedural and object-oriented programming. The reusability of code, classes, and objects allow us to model the real world. Python has ways of combining data and functions into one form or class.

Dynamically Typed: Until the code is run (i.e. executed) Python does not know the type of variable. It allocates the data type automatically during execution—no need to worry the programmer about declaring variables and their data types.

Free and Open-Source: Python comes under an open-source approved by OSI. That allows it to be used and distributed freely. You can access and edit the source code and even distribute your Python version.

Disadvantages of Python

There are some disadvantages to using Python. These are summarised below:

Difficulty to use other programming languages: Python users become very used to the functionality and vast libraries of Python, and so they face difficulties understanding or working in other programming languages.

Weak in Mobile Computing: The presence of Python on many desktop and server platforms is noticeable, but the language is regarded as a weak mobile computing language. This is the reason why very few smartphone apps are integrated into it.

Gets Slow: Python needs to execute with the help of an interpreter instead of the compiler which tends to cause it to slow down. However, for many web applications, it is fast and efficient as well.

Not Memory Efficient: The Python programming language utilises a large amount of memory. This could be a drawback if memory optimisation is preferred when designing applications.

Run-time Errors and design constraints: The Python language is dynamically typed and it has several design constraints that some Python developers have documented. More testing time appears to be needed to run Python programs, and when the applications eventually run time errors may appear.

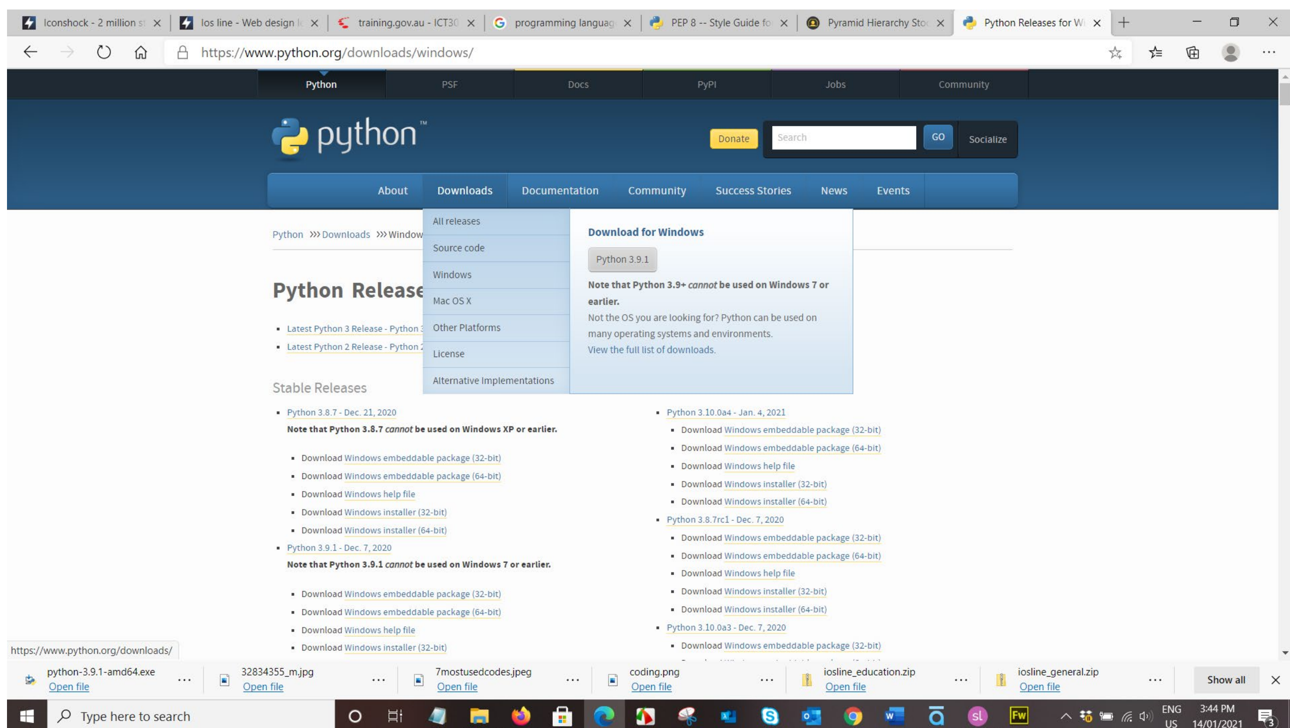
Design Restrictions: Python is typed dynamically. This means you do not have to specify the type of a variable when you write the code; it can cause errors in run-time.

Underdeveloped Database Access Layers: Compared with the current technologies such as JDBC and ODBC, the access layer for the Python database is found to be somewhat underdeveloped and basic. This restricts the use of Python in organisations that require complex legacy data to communicate smoothly.

How to Install Python on Windows

Install Python

To use python on your Windows operating system, open a browser window and navigate to the Download page for Windows at python.org.

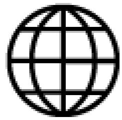


Step 1: Download

Click on the link for the New Python 3 Update-Python 3.x.x below the top heading which says Python Releases for Windows. (As of this writing, Python 3.9.1 is latest, most stable version.)

Scroll down, and you have choices to select from either the 64-bit executable Windows x86-64 installer or the 32-bit executable Windows x86 installer.

Save the application in a folder or it may automatically download to your Download folder.

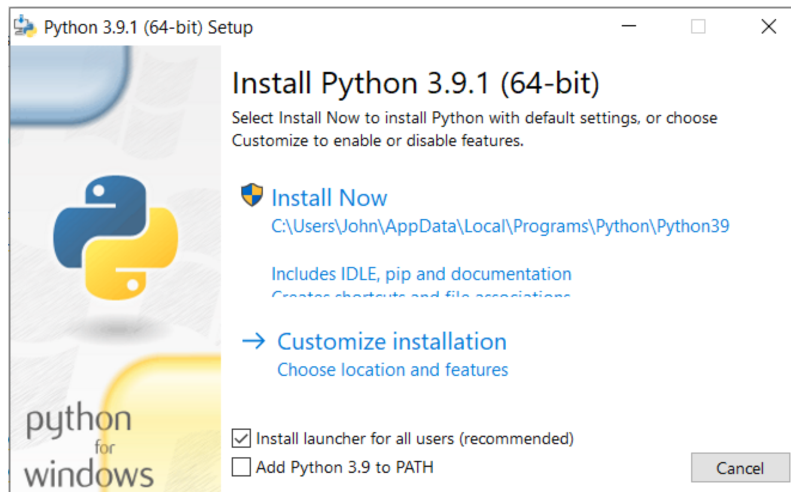


Python Download *(https://www.python.org/)*

Step 2: Installation

To install Python, simply click the downloaded file, the installation process will begin.

A pop-up box will appear for you to follow instructions to install Python to your computer. You must ensure you download Python for the appropriate version of the Windows 32-bit or 64-bit.



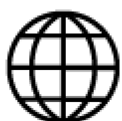
Follow the instructions and install the Python in your computer.

Writing your First Python Program

To write the “Hello, World!” program, let us open a command-line text editor such as notepad, pico, nano etc. and create a new file.

The text editors are used to write all the Python code/programming. You can write Python in any text editor. However, there are some text editors those are especially designed for the writing the Python code they are called IDE (Integrated Development Environment).

We will be using PyCharm for writing Python code in this tutorial. You will be able to install PyCharm from https://www.jetbrains.com/pycharm/.



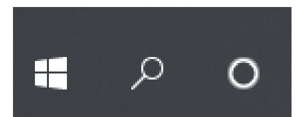
Pycharm - Integrated Development Environment

(https://www.jetbrains.com/pycharm/)

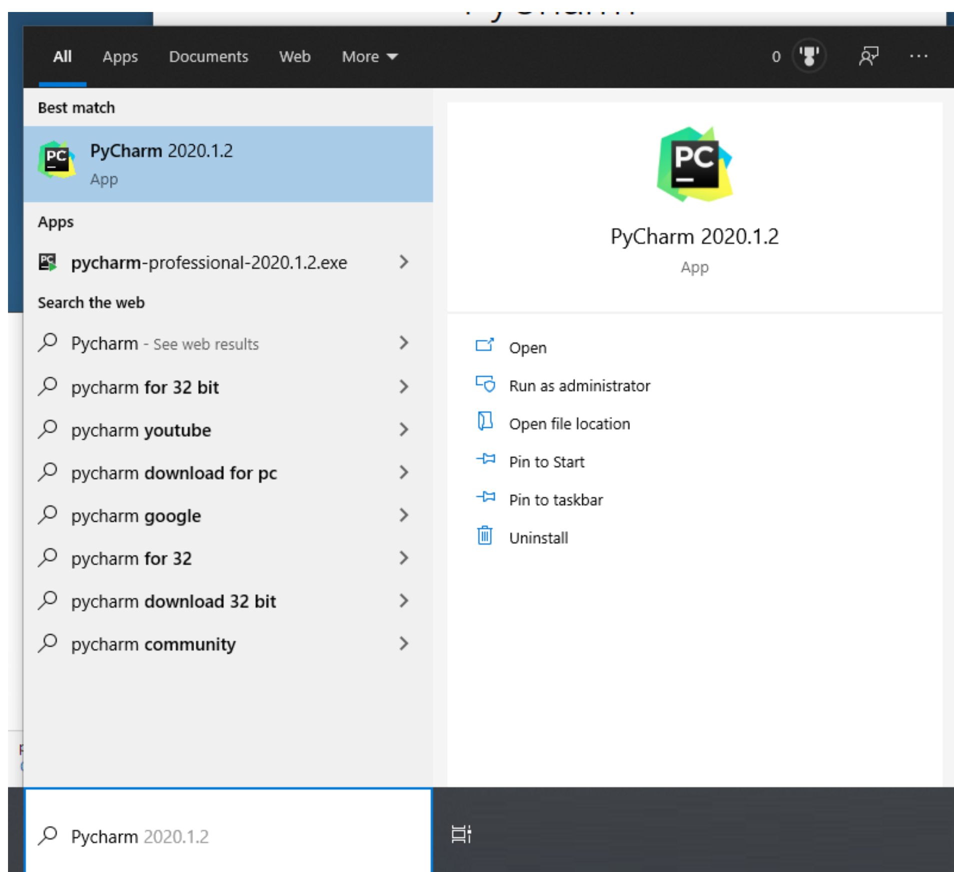
You will have options of selecting either a professional (free trial) or community-based, free software application tool version. The community-based version has everything that you need to start using Python language. You must run and install the software on your computer. You are at this stage, ready to start writing your first Python program.

Step 1: Open PyCharm IDE

The first step is to open up PyCharm IDE software.



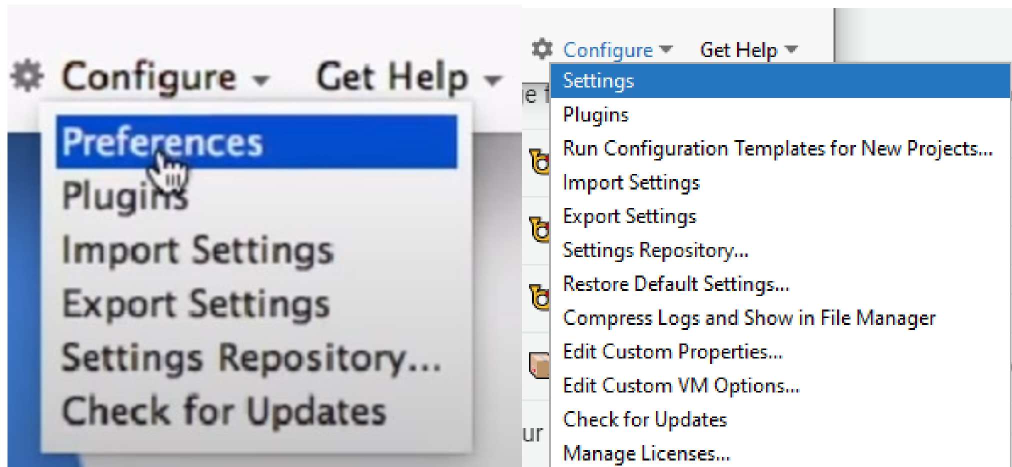
Go to windows Search icon next to start button and in the search bar type **PyCharm** and click on the PyCharm application name when it appears.



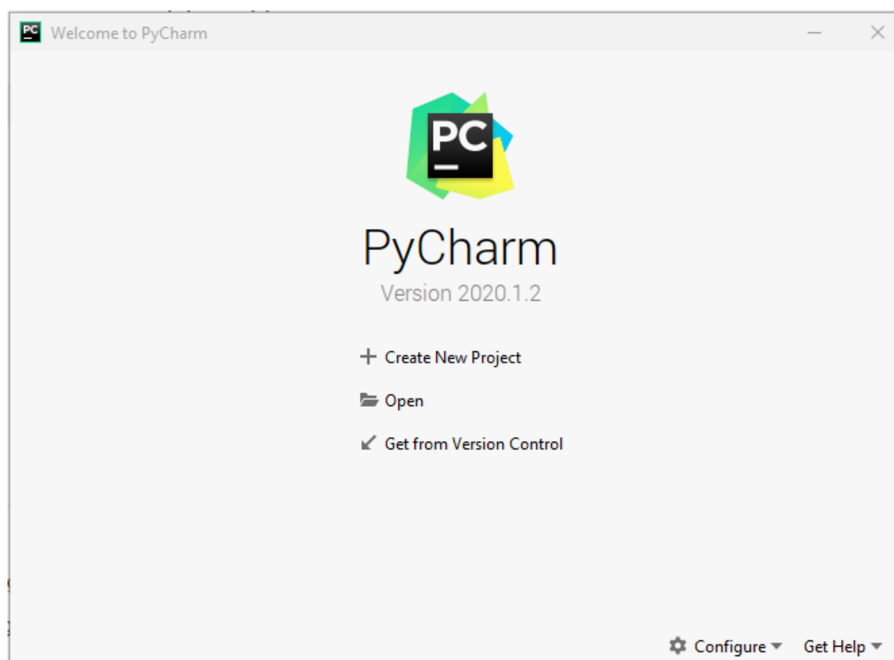
Step 2: Create a new project

This step involves creating a new project using the Python language.

Before you create a new project, you have the options to configure and change the theme of the PyCharm – Python IDE software application. The preferences option appears under the "configure" link on Mac and "settings" link in the Windows operating system.

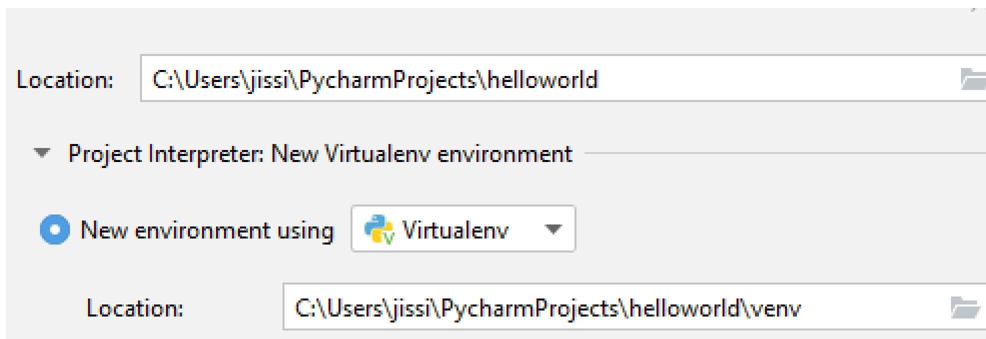


If you do not want to change the theme, you can select "create new project" from the dialogue box.

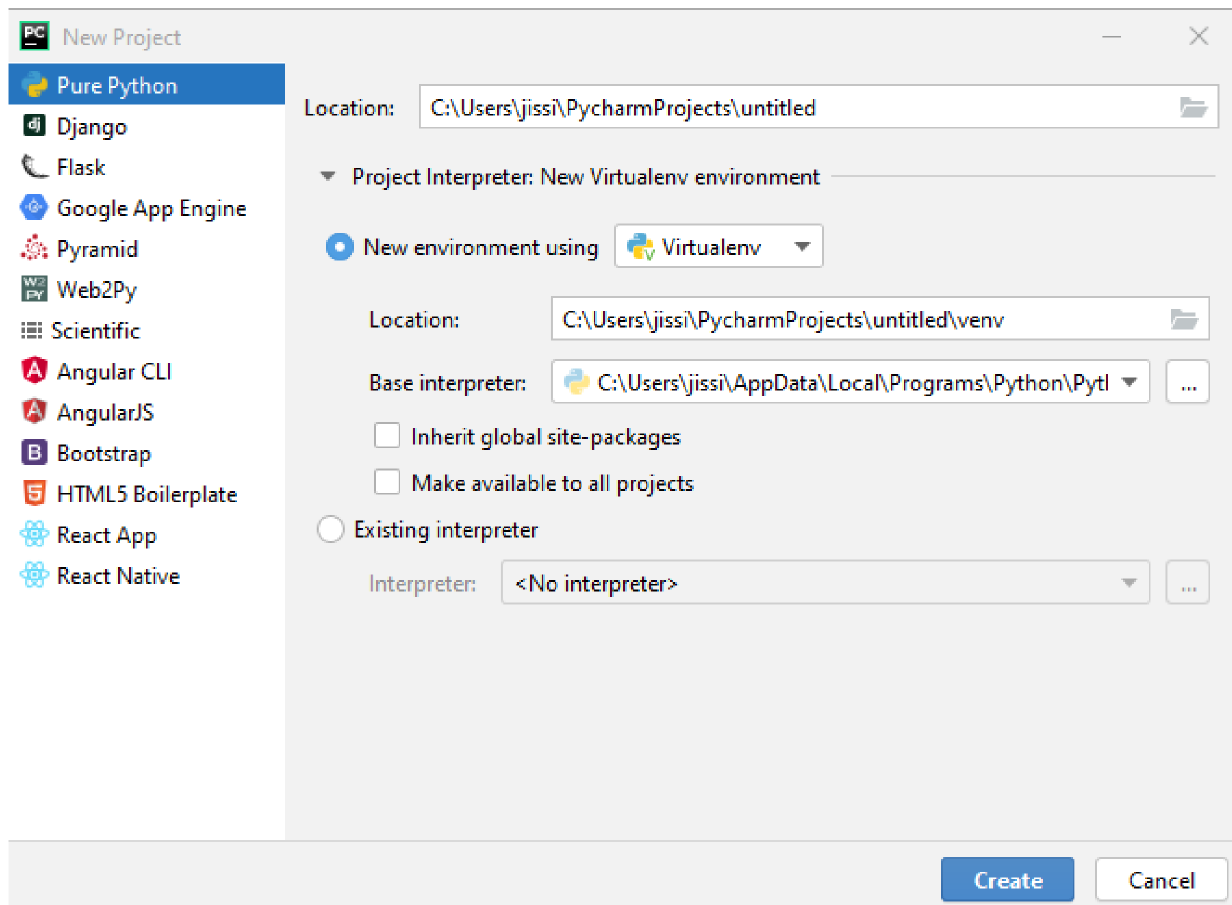


When you click on the "create new project", a new window will appear where you can name your project in the location tab.

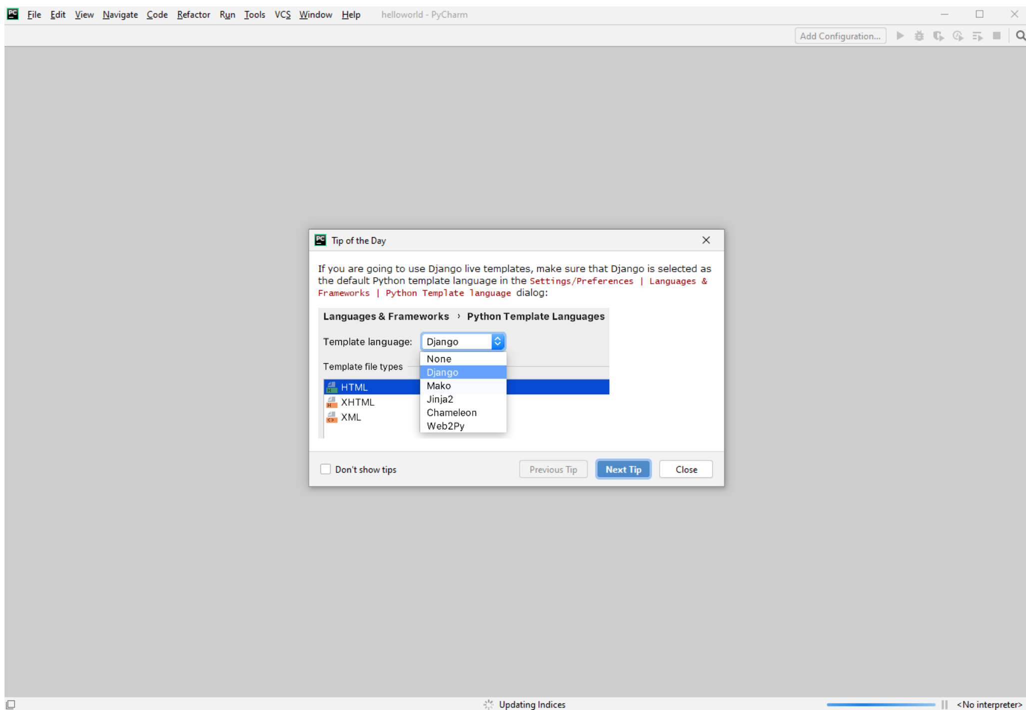
Write on top of intitled the name of the project you would like to assign to your first Python program. We are calling it hello world, for example, so the name given is "helloworld" as demonstrated in the image below.



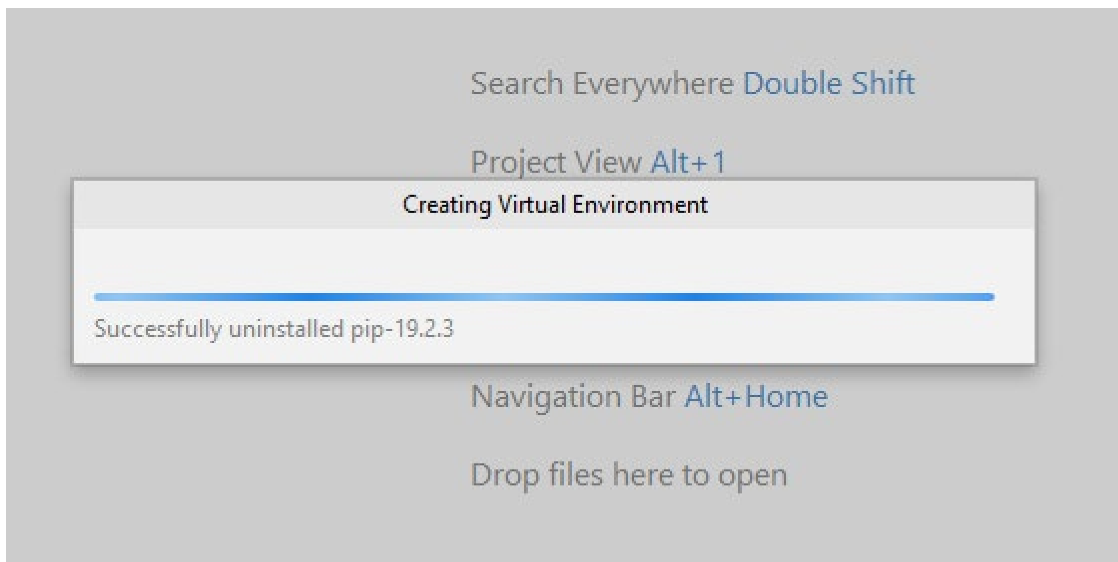
Note: You must make sure your base interpreter is showing Python 3 not 2 to interpret the Python code. On the MAC computer, you would notice multiple options as Python version 2 is pre-installed with the operating system.



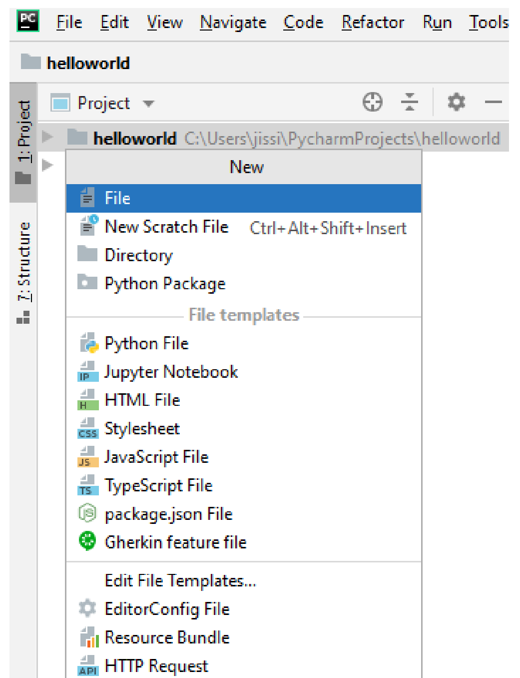
Click create after the name is assigned to your first project. The below screen will appear if you are correctly following the instructions.



This step may take some time as the IDE is creating a virtual environment to run your Python program.



Click on the new main menu and click on Python File in the menu under "file templates" section.



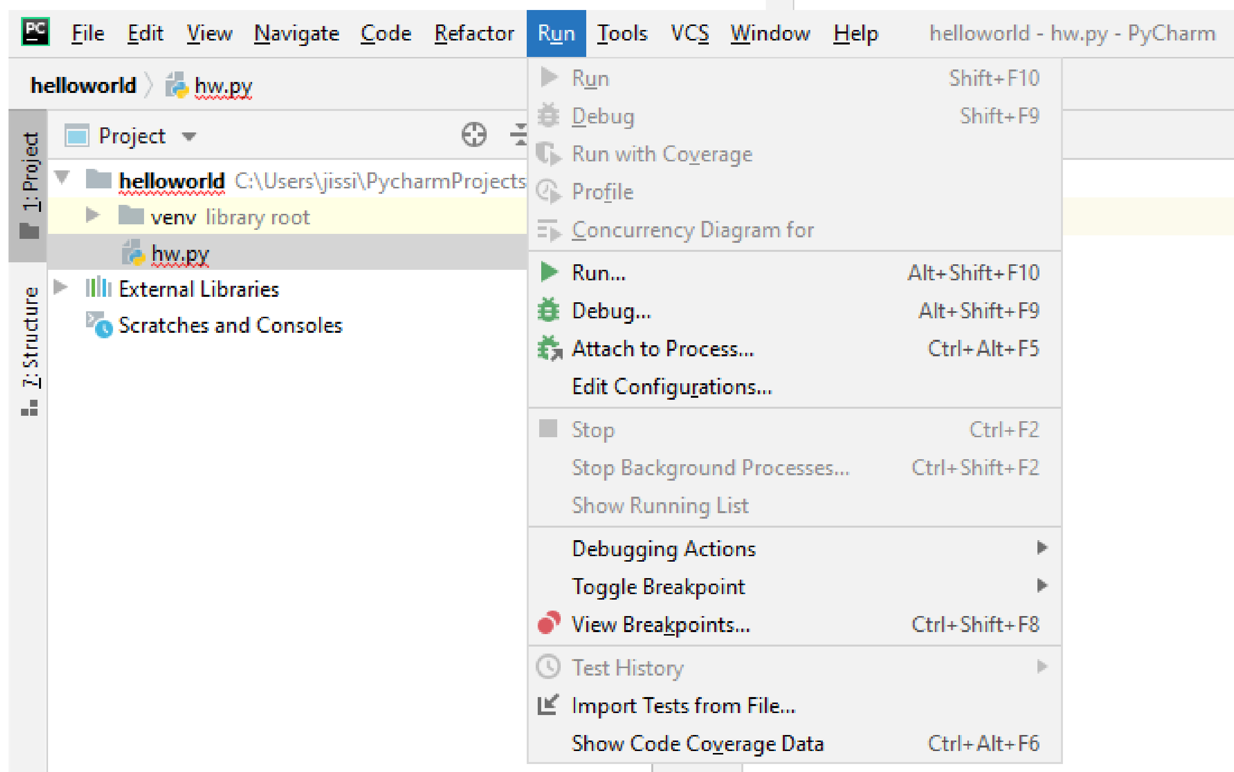
Give the file a name. We are giving it the name of "hw". Write the name and press enter on your computer, and you would notice a hw.py file opens up in the editor.

This is the moment when we can start typing our Python code. The first program we have selected is "hello world" to print a message out on the screen.

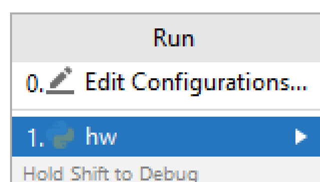
Wait for the text file opens up in the terminal window, we will then type out our first code:

```
print ("hello world")
```

You should then click on "Run" main menu and select the option of "Run" the code.



A dialogue box will appear for you to select how would you like to run the program. Select the application option.



You would notice the outcome in the console window.

Apply Language Syntax and Layout

It is now time to start to understand the syntax of Python. Various elements of the software are outlined below:

'Print()' It is a function which tells the machine to execute an action. We know this is a function since it uses parentheses. The print () function provides the necessary information to Python to print the output (specified message) in the parenthesis that we put in. It will get output to the current terminal window by default. The specified message can be a string or any other object.

Many features, such as the 'print()' function, are integrated and built-in features that are used by default in Python. Such integrated functions are still available for us to use in the programs we design and build. We can also describe our own functions through other elements, which we construct ourselves.

There is a series of characters within the parentheses of the print() Hello, World! — that's expressed using the quotation marks. A string is described as ay characters within quotation marks.

Apply basic language syntax rules

Before we start learning Python, let's get familiar with the common Python programming environment.

The Python programming language syntax is the collection of rules and guidelines that describe how to write and interpret a Python program (by both the runtime system and by human readers). The language used in Python has many striking similarities with other languages such as Perl, Java, C++ and C. However, there are also some significant differences between these programming languages.

Python Syntax Rules

1. Python is case sensitive. Hence a variable with name WeAreinClass is not same as weareinclass.
2. Python uses forward slashes for path specification. Therefore if you are operating with a file, in the case of Windows OS, the default file path will have backward slashes that you will have to convert to forward slashes to make them work in the python script.

Let's take an example, Window's path C:\folderA\folderB relative to the Python program path should be C:/folderA/folder

There is no command terminator in Python, which implies no use of semicolon; or anything.

Therefore if you want something to be printed as output, all you would write:

```
print ("Hello, World!")
```

This is the first python program too that we just shared with you. Only using one simple single-line statement.

3. Only one executable statement should be written in one line, and the line change in Python will function as a command terminator.

   You should use a semicolon to write two separate executable statements in a single line; to differentiate the instructions.

   Example:

```
print ("Hello, World!") ; print ("This is the second statement")
```

4. The use of single quotes '', double quotes "" and even triple quotes ''' """ in the Python programming language demonstrate the string literals.

```
word = 'world'

sentence = "This is a one line statement."

para = """This is a paragraph

which has multiple lines using the double or triple quotes """
```

5. You can write comments in the Python programming language using a hash # symbol at the start in your program. The comment is used to provide necessary information on what sections of the code should be ignored while the python script is executed.

```
# this is an example of using comment

print ("Hello, World!")

# this is an example of

# multiline comment
```

6. Line Continuation: The backslash \ at the end of each line is used to write a code in multiline without confusing the python interpreter. This is used explicitly to denote line continuation. Let's consider an example,

```
sum =  321 + \
```

```
654 + \
976
```

Note: Expressions enclosed in ( ), [ ] or { } brackets do not require the use of a backward slash for line continuation. Let's consider an example,

```
vowels = ['a', 'e', 'i',
        'o', 'u']
```

7. The Python programming language ignores the blank lines in between a program.

8. Code Indentation: A code block (a function's body, loop, and so on.) starts with the indentation and ends with the first unindented line. In the other programming languages, such as Java, C or C++, usually, the curly brackets { } are used to define a code block. However, these curly brackets are not used in the python programming language. The python programming language use indentation for this purpose.

   The amount of indentation is up to the programmer; however, in the block, it is required to be consistent. Four white spaces are generally used for indentation and preferred over tabs.

   Recommendation: You should use tab for indentation even though you may use spaces as well. The tabs are used to be consistent with providing the right amount of indentation for a single block of code.

```
if True:
print ("Yes, I am writing in the block");
# the above statement will not be
# considered outside the if block
```

So, what is the correct way to write the code above,

```
if True:
    # this is inside if block
    print ("Yes, I am writing in the if block")
```

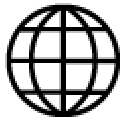
Please note, the following code will give you an error, as the statements are not indented using the right amount:

```
if True:
    # this is inside if block
    print ("Yes, I am writing in the block")
    # this will give error
        print ("you should as well")
```

Once again, the correct method to do so is to keep all the statements of a particular code block consistent with providing the right amount of indentation.

```
if True:
    # this is inside if block
    print ("Yes, I am writing in the block")
    print ("you should as well")
```

These are some of the basic rules and guidelines in the Python programming language.



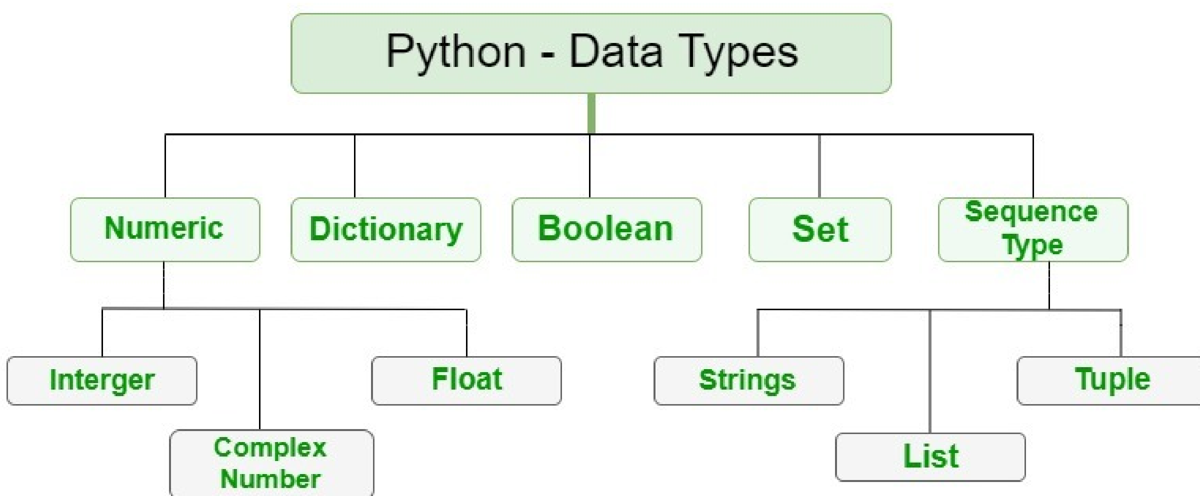
Python Syntax *(https://www.w3resource.com/python/python-syntax.php)*

Create code using data types, operators and expressions

Data Types in Python

Data types are explained as classification or categorisation of a particular kind of data item. The values define data types it can take that decides which operations on that data can be performed. The commonly used data types are numeric, non-numeric, and Boolean (true/false). Because every programming language has its own classification that largely reflects their programming philosophy

Within such classification or categorisation, Python has the following data types built in by default:



Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Setting the specific data type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called an operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Let us have a look on all operators one by one.

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The table below lists the arithmetic operator:

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*****	Multiplication Operator
/	Division Operator
%	Divides left-hand operand by right-hand operand and returns the remainder
**** Exponent**	Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) −

Example: Arithmetic Operator

Assume variable a hold 21, and variable b holds 10, then...

```
#!/usr/bin/python

a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c

c = a % b
print "Line 5 - Value of c is ", c

a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c

a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```

When you run the program, the output will be:

Line 1 - Value of c is 31

Line 2 - Value of c is 11

Line 3 - Value of c is 210

Line 4 - Value of c is 2

Line 5 - Value of c is 1

Line 6 - Value of c is 8

Line 7 - Value of c is 2

Unary Operators

Unary operator performs operation on only one operand.

Addition

The + operator in Python can be utilized in a unary form. The unary structure implies character, restoring the same value as its operand.

Example:

Prototype	Example
+ (int) -> int	+4 returns the result 4
+ (float) -> float	+4.0 returns the result 4.0
+ (complex) -> complex	+4j returns the result 4j

Python Code:

```
print +3
print +3.0
print +3j
```

Output:

```
3
3.0
3j
```

Subtraction

The – operator in Python can be utilized in a unary form. The unary structure implies character, restoring the same value as its operand. The unary structure implies they negate, restoring the nullified an incentive as its operand: zero to zero, positive to negative, and negative to positive.

Example:

Prototype	Example
– (int) -> int	-4 returns the result -4
– (float) -> float	-4.0 returns the result -4.0
– (complex) -> complex	-4j returns the result -4j

Python Code:

```
print -4
print -4.0
print -4j
```

Output:

```
 -4
-4.0
-4j
```

Multiplication

The * operator in Python can be utilized distinctly in the paired structure, which implies increase, restoring an outcome that is the standard arithmetic product result of its operands.

Example:

Prototype	Example
*** (int, int) -> int**	4* 5 returns the result 20
*** (float, float) -> float**	4.0 * 5.0 returns the result 20.0
*** (complex, complex) -> complex**	4j * 5j returns the result (-20+0j)

Python Code:

```
print 3*5
print 3.0*5.0
print 3j*5j
```

Output:

```
 15
15.0
(-15+0j)
```

Division

The / Operator(cut) in Python can be utilized distinctly in the parallel structure, which implies division, restoring an outcome that is the standard number juggling the remainder of its operands: left operand partitioned by the right operand.

Example:

Prototype	Example
/ **(int, int) -> int**	4 / 5 returns the result 0.8
/ **(float, float) -> float**	4.0 / 5.0 returns the result 0.8
/ **(complex, complex) -> complex**	4j / 5j returns the result (.08+0j)

Python Code:

```
print 3/5
print 3.0/5.0
print 3j/5j
```

Output:

```
.6
0.6
(0.08+0j)
```

Floor Division

The // operator (twofold slice) in Python can be utilized as it were in the twofold structure, which additionally implies division, yet restoring a vital outcome of the standard number juggling the remainder of its operands: left operand partitioned by the right operand.

Example:

Prototype	Example
// (int, int) -> int	3 // 5 returns the result 0
// (float, float) -> float	3.0 // 5.0 returns the result 0.0

Python Code:

```
print 3//5
print 3.0//5.0
```

Output:

```
0
0.0
```

Modulo

The % operator in Python can be utilized distinctly in the parallel structure, which implies the leftover portion after the left operand partitioned by the correct operand.

Model Example:

Prototype	Example
% (int, int) -> int	3%4 returns the result 0
% (float, float) -> float	3.0%5.0 returns the result 3.0

Python Code:

```
print 3%5
print 3.0%5.0
```

Output:

```
 3
3.0
```

Power

The ** operator in Python can be utilized distinctly in the double structure, which implies power restoring an outcome that is the left operand raised to the intensity of the correct operand.

Example:

Prototype	Example
**** (int, int) -> int**	3 ** 5 returns the result 243
**** (float, float) -> float**	3.0 ** 5.0 returns the result 243.0
**** (complex, complex) -> complex**	3j ** 5j returns the result (0.000273200847641 43374-0.000275795258093 76897j

Python Code:

```
print 3**5
print 3.0**5.0
print 3j**5j
```

Output:

```
 243
243.0
(0.000273200847641-0.000273200847641j
```

Boolean Operators

In Python, and, or and not are Boolean operators. With Boolean operators, we perform legitimate tasks. These are regularly utilized with if and while keywords.

Python Code:

```
print (True and True)
print (True and False)
print (False and True)
print (False and False)
```

Output:

True

False

False

False

Relational Operator

Relational operators used for comparing values. It returns a Boolean value.

Symbol	Meaning
<	Less than
<=	Less than equal to
>	Greater than
>=	Greater than equal to
==	Equal to
!=	Not equal to

Python Code:

```
print 3 < 4
print 4 == 3
print 4 >= 3
```

As we previously referenced, the social administrators return Boolean qualities: True or False.

Output:

True

False

True

Python Object Identity Operators

The object identity operators consist of is and not is, it checks if its operators are a similar item.

Let's take an example.

```
print (Equals == Equals)
print (Equals is Equals)
print (Equals is Equals)
print ([] == [])
print ([] is [])
print ("Python" is "Python")
```

Output:

True

True

True

True

False

True

Python Bitwise Operators

Decimal numbers are normal for people. Double numbers are local to PCs. Double, octal, decimal or hexadecimal images are just documentation of a similar number. Bitwise administrators work with bits of a double number.

Symbol	Meaning
~	negation
^	exclusive or
&	and
|	or
<<	left shift
>>	right shift

The bitwise negation operator makes the changes to every 1 to 0 and 0 to 1.

Python Code:

```
print ~7
print ~-8
```

Output:

-8

7

The administrator returns all bits of a number 7.

Expressions

A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value.

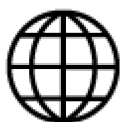
Example (save as expression.py):

```
length = 5
breadth = 2
area = length * breadth
print('Area is', area)
print('Perimeter is', 2 * (length + breadth))
Output:
$ python expression.py
Area is 10
Perimeter is 14
```

How it works

The length and breadth of the rectangle are stored in variables by the same name. We use these to calculate the area and perimeter of the rectangle with the help of expressions. We store the result of the expression length * breadth in the variable area and then print it using the print function. In the second case, we directly use the value of the expression 2 * (length + breadth) in the print function.

Notice how Python pretty-prints the output. Even though we have not specified a space between 'Area is' and the variable area, Python, puts it for us so that we get a nice clean output and the program is much more readable this way (since we don't need to worry about spacing in the strings we use for output). This is an example of how Python makes life easy for the programmer.



Python Expressions *(https://docs.python.org/3/reference/expressions.html)*

Apply variables and variable scope

What are variables in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

What is the variable scope in Python?

The scope of a variable in python is that part of the code where it is visible. To refer to it, you don't need to use any prefixes then. Let's take an example, but before let's revise python Syntax. Also, the duration for which a variable is alive is called its 'lifetime'.

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

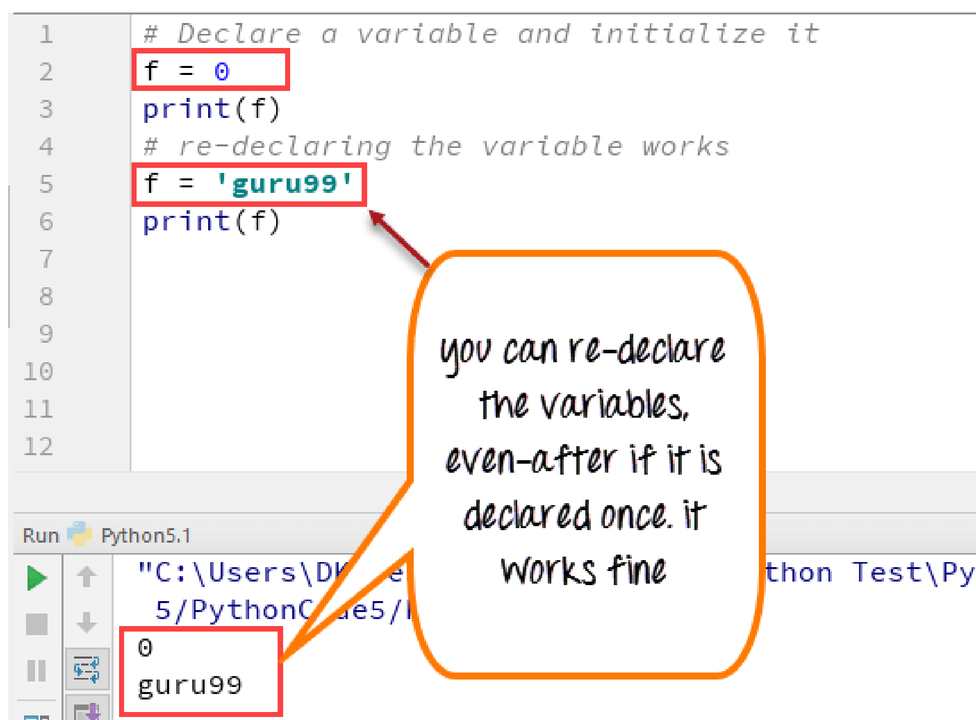
```
a=100

print (a)
```

Re-declare a Variable

You can re-declare the variable even after you have declared it once.

Here we have variable initialized to f=0.

Later, we re-assign the variable f to value "guru99"



Python 2 Example

```
# Declare a variable and initialize it
f = 0
print f
# re-declaring the variable works
f = 'guru99'
print f
```

Python 3 Example

```
# Declare a variable and initialize it
f = 0
print(f)
# re-declaring the variable works
f = 'guru99'
print(f)
```

Concatenate Variables

Let's see whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as a string, Python requires declaring the number as string otherwise it will show a TypeError

```
6   print(f)
7
8
9   #ERROR: different types cannot be combined
10  print("guru"+99)
11
12
```

```
Run  Python5.1
"C:\Users\DK\Desktop\Python code\Pytho
 5/PythonCode5/Python5.1.py"
Traceback (most recent call last):
0
  File "C:/Users/DK/Desktop/Pytho  code
guru99
    print("guru"+99)
TypeError: must be str, not int
```

it shows type error as #"99" is not declared as string

For the following code, you will get undefined output -

a="Guru"

b = 99

print a+b

Once the integer is declared as string, it can concatenate both "Guru" + str("99")= "Guru99" in the output.

a="Guru"

b = 99

print(a+str(b))

Local & Global Variables

In Python when you want to use the same variable for rest of your program or module, you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable.

Let's understand this difference between a local and global variable with the below program.

Variable "f" is global in scope and is assigned value 101 which is printed in output

Variable f is again declared in function and assumes local scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier

Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

Python5.2.py

```
1   # Declare a variable and initialize it
2   f = 101
3   print(f)
4
5   # Global vs. local variables in functions
6   def someFunction():
7   # global f
8       f = 'I am learning Python'
9       print(f)
10
11  someFunction()
12  print(f)
13
```

f is a local variable declared inside the function.

Run Python5.2

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 5\PythonCode5\PythonCode5/Python5.2.py"
101
I am learning Python
101
```

Python 2 Example

```
# Declare a variable and initialize it
f = 101
print f
# Global vs. local variables in functions
def someFunction():
# global f
    f = 'I am learning Python'
    print f
someFunction()
print f
```

Python 3 Example

```
# Declare a variable and initialize it
f = 101
print(f)
# Global vs. local variables in functions
def someFunction():
# global f
    f = 'I am learning Python'
    print(f)
someFunction()
print(f)
```

Using the keyword **global**, you can reference the global variable inside a function.

Variable "f" is global in scope and is assigned value 101 which is printed in output

Variable f is declared using the keyword global. This is NOT a local variable, but the same global variable declared earlier. Hence when we print its value, the output is 101

We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists. At line 12, when we again, print the value of "f" is it displays the value "changing global variable."

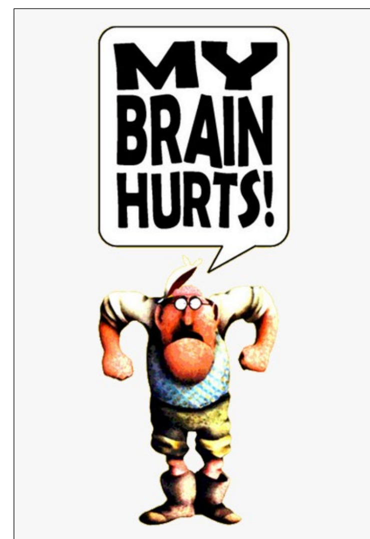
```
1   f = 101;
2   print(f)        1
3
4   # Global vs.local variables in functions
5   def someFunction():
6       global f
7       print(f)        2
8       f = "changing global variable"
9
10  someFunction()
11  print(f)        3
12
13
14
```

We are now accessing and changing the global variable f.

```
someFunction()
Run  Python5.3
"C:\Users\DK\Desktop\Python code\Python Test\Python 5\Pytho
 5/PythonCode5/Python5.3.py"
101
101
changing global variable
```

Python 2 Example

```
f = 101;
print f
# Global vs.local variables in functions
def someFunction():
  global f
  print f
  f = "changing global variable"
someFunction()
print f
```



Python 3 Example

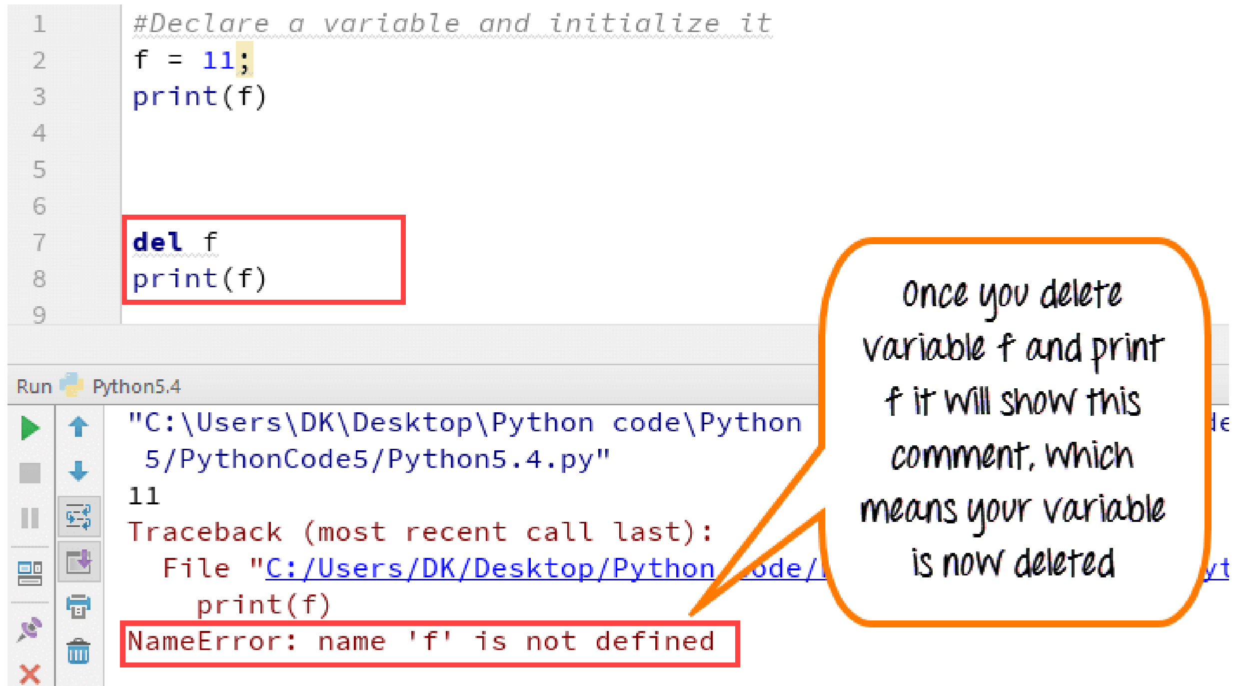
```
f = 101;
print(f)
# Global vs.local variables in functions
def someFunction():
  global f
  print(f)
  f = "changing global variable"
```

```
someFunction()
print(f)
```

Delete a variable

You can also delete variable using the command del "variable name".

In the example below, we deleted variable f, and when we proceed to print it, we get error "variable name is not defined" which means you have deleted the variable.




```
f = 11;
print(f)
del f
print(f)
```

Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

Let's see an example of how a global variable is created in Python.

Example 1: Create a Global Variable

```
x = "global"
def foo():
   print("x inside:", x)
foo()
print("x outside:", x)
```

Output

x inside: global

x outside: global

In the above code, we created x as a global variable and defined a foo() to print the global variable x. Finally, we call the foo() which will print the value of x.

What if you want to change the value of x inside a function?

```
x = "global"


def foo():
    x = x * 2
    print(x)


foo()
```

Output

UnboundLocalError: local variable 'x' referenced before assignment

The output shows an error because Python treats x as a local variable and x is also not defined inside ‘foo()’.

To make this work, we use the global keyword.

Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.

Example 2: Accessing local variable outside the scope

```
def foo():
    y = "local"
foo()
print(y)
```

Output

NameError: name 'y' is not defined

The output shows an error because we are trying to access a local variable y in a global scope whereas the local variable only works inside foo() or local scope.

Let's see an example of how a local variable is created in Python.

Example 3: Create a Local Variable

Normally, we declare a variable inside the function to create a local variable.

```
def foo():
    y = "local"
    print(y)

foo()
```

Output

local

Let's take a look at where x was a global variable, and we wanted to modify x inside foo().

Global and local variables

Here, we will show how to use global variables and local variables in the same code.

Example 4: Using Global and Local variables in the same code

```
x = "global "

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

Output

global global

local

In the above code, we declare x as a global and y as a local variable in the foo(). Then, we use multiplication operator * to modify the global variable x, and we print both x and y.

After calling the foo(), the value of x becomes global global because we used the x * 2 to print two times global. After that, we print the value of local variable y, i.e. local.

Example 5: Global variable and Local variable with the same name

```
x = 5

def foo():
    x = 10
    print("local x:", x)


foo()
print("global x:", x)
```

Output

```
local x: 10
global x: 5
```

In the above code, we used the same name x for both global variable and local variable. We get a different result when we print the same variable because the variable is declared in both scopes, i.e. the local scope inside 'foo()' and global scope outside 'foo()'.

When we print the variable inside 'foo()', it outputs local x: 10. This is called the local scope of the variable.

Similarly, when we print the variable outside the 'foo()', it outputs global x: 5. This is called the global scope of the variables.

Nonlocal Variables

Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Let's see an example of how a global variable is created in Python.

We use nonlocal keywords to create nonlocal variables.

Example 6: Create a nonlocal variable

```
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)
```

```
    inner()
    print("outer:", x)

outer()
```

Output

    inner: nonlocal
    outer: nonlocal

In the above code, there is a nested 'inner()' function. We use nonlocal keywords to create a nonlocal variable. The 'inner()' function is defined in the scope of another function 'outer()'.

Use program library functions

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some of the optional components.

Click on the links below to know more about library functions:

1. Introduction
    - Notes on availability
2. Built-in Functions
3. Built-in Constants
    - Constants added by the site module
4. Built-in Types
    - Truth Value Testing
    - Boolean Operations — and, or, not
    - Comparisons
    - Numeric Types — int, float, complex
    - Iterator Types
    - Sequence Types — list, tuple, range
    - Text Sequence Type — str
    - Binary Sequence Types — bytes, bytearray, memoryview
    - Set Types — set, frozenset
    - Mapping Types — dict
    - Context Manager Types
    - Other Built-in Types
    - Special Attributes
5. Built-in Exceptions
    - Base classes
    - Concrete exceptions
    - Warnings
    - Exception hierarchy
6. Text Processing Services
    - string — Common string operations
    - re — Regular expression operations
    - difflib — Helpers for computing deltas
    - textwrap — Text wrapping and filling
    - unicodedata — Unicode Database

    - stringprep — Internet String Preparation
    - readline — GNU readline interface
    - rlcompleter — Completion function for GNU readline
7. Binary Data Services
    - struct — Interpret bytes as packed binary data
    - codecs — Codec registry and base classes
8. Data Types
    - datetime — Basic date and time types
    - calendar — General calendar-related functions
    - collections — Container datatypes
    - collections.abc — Abstract Base Classes for Containers
    - heapq — Heap queue algorithm
    - bisect — Array bisection algorithm
    - array — Efficient arrays of numeric values
    - weakref — Weak references
    - types — Dynamic type creation and names for built-in types
    - copy — Shallow and deep copy operations
    - pprint — Data pretty printer
    - reprlib — Alternate repr() implementation
    - enum — Support for enumerations
9. Numeric and Mathematical Modules
    - numbers — Numeric abstract base classes
    - math — Mathematical functions
    - cmath — Mathematical functions for complex numbers
    - decimal — Decimal fixed point and floating point arithmetic
    - fractions — Rational numbers
    - random — Generate pseudo-random numbers
    - statistics — Mathematical statistics functions
10. Functional Programming Modules
    - itertools — Functions creating iterators for efficient looping
    - functools — Higher-order functions and operations on callable objects
    - operator — Standard operators as functions
11. File and Directory Access
    - pathlib — Object-oriented filesystem paths
    - os.path — Common pathname manipulations
    - fileinput — Iterate over lines from multiple input streams
    - stat — Interpreting stat() results
    - filecmp — File and Directory Comparisons
    - tempfile — Generate temporary files and directories
    - glob — Unix style pathname pattern expansion
    - fnmatch — Unix filename pattern matching
    - linecache — Random access to text lines
    - shutil — High-level file operations
12. Data Persistence
    - pickle — Python object serialization
    - copyreg — Register pickle support functions

  - shelve — Python object persistence
  - marshal — Internal Python object serialization
  - dbm — Interfaces to Unix “databases”
  - sqlite3 — DB-API 2.0 interface for SQLite databases
13. Data Compression and Archiving
  - zlib — Compression compatible with **gzip**
  - gzip — Support for **gzip** files
  - bz2 — Support for **bzip2** compression
  - lzma — Compression using the LZMA algorithm
  - zipfile — Work with ZIP archives
  - tarfile — Read and write tar archive files
14. File Formats
  - csv — CSV File Reading and Writing
  - configparser — Configuration file parser
  - netrc — netrc file processing
  - xdrlib — Encode and decode XDR data
  - plistlib — Generate and parse Mac OS X .plist files
15. Cryptographic Services
  - hashlib — Secure hashes and message digests
  - hmac — Keyed-Hashing for Message Authentication
  - secrets — Generate secure random numbers for managing secrets
16. Generic Operating System Services
  - os — Miscellaneous operating system interfaces
  - io — Core tools for working with streams
  - time — Time access and conversions
  - argparse — Parser for command-line options, arguments and sub-commands
  - getopt — C-style parser for command line options
  - logging — Logging facility for Python
  - logging.config — Logging configuration
  - logging.handlers — Logging handlers
  - getpass — Portable password input
  - curses — Terminal handling for character-cell displays
  - curses.textpad — Text input widget for curses programs
  - curses.ascii — Utilities for ASCII characters
  - curses.panel — A panel stack extension for curses
  - platform — Access to underlying platform’s identifying data
  - errno — Standard errno system symbols
  - ctypes — A foreign function library for Python
17. Concurrent Execution
  - threading — Thread-based parallelism
  - multiprocessing — Process-based parallelism
  - multiprocessing.shared_memory — Provides shared memory for direct access across processes
  - The concurrent package
  - concurrent.futures — Launching parallel tasks
  - subprocess — Subprocess management

- sched — Event scheduler
- queue — A synchronized queue class
- _thread — Low-level threading API
- _dummy_thread — Drop-in replacement for the _thread module
- dummy_threading — Drop-in replacement for the threading module

18. contextvars — Context Variables
    - Context Variables
    - Manual Context Management
    - asyncio support
19. Networking and Interprocess Communication
    - asyncio — Asynchronous I/O
    - socket — Low-level networking interface
    - ssl — TLS/SSL wrapper for socket objects
    - select — Waiting for I/O completion
    - selectors — High-level I/O multiplexing
    - asyncore — Asynchronous socket handler
    - asynchat — Asynchronous socket command/response handler
    - signal — Set handlers for asynchronous events
    - mmap — Memory-mapped file support
20. Internet Data Handling
    - email — An email and MIME handling package
    - json — JSON encoder and decoder
    - mailcap — Mailcap file handling
    - mailbox — Manipulate mailboxes in various formats
    - mimetypes — Map filenames to MIME types
    - base64 — Base16, Base32, Base64, Base85 Data Encodings
    - binhex — Encode and decode binhex4 files
    - binascii — Convert between binary and ASCII
    - quopri — Encode and decode MIME quoted-printable data
    - uu — Encode and decode uuencode files
21. Structured Markup Processing Tools
    - html — HyperText Markup Language support
    - html.parser — Simple HTML and XHTML parser
    - html.entities — Definitions of HTML general entities
    - XML Processing Modules
    - xml.etree.ElementTree — The ElementTree XML API
    - xml.dom — The Document Object Model API
    - xml.dom.minidom — Minimal DOM implementation
    - xml.dom.pulldom — Support for building partial DOM trees
    - xml.sax — Support for SAX2 parsers
    - xml.sax.handler — Base classes for SAX handlers
    - xml.sax.saxutils — SAX Utilities
    - xml.sax.xmlreader — Interface for XML parsers
    - xml.parsers.expat — Fast XML parsing using Expat
22. Internet Protocols and Support
    - webbrowser — Convenient Web-browser controller

- cgi — Common Gateway Interface support
- cgitb — Traceback manager for CGI scripts
- wsgiref — WSGI Utilities and Reference Implementation
- urllib — URL handling modules
- urllib.request — Extensible library for opening URLs
- urllib.response — Response classes used by urllib
- urllib.parse — Parse URLs into components
- urllib.error — Exception classes raised by urllib.request
- urllib.robotparser — Parser for robots.txt
- http — HTTP modules
- http.client — HTTP protocol client
- ftplib — FTP protocol client
- poplib — POP3 protocol client
- imaplib — IMAP4 protocol client
- nntplib — NNTP protocol client
- smtplib — SMTP protocol client
- smtpd — SMTP Server
- telnetlib — Telnet client
- uuid — UUID objects according to **RFC 4122**
- socketserver — A framework for network servers
- http.server — HTTP servers
- http.cookies — HTTP state management
- http.cookiejar — Cookie handling for HTTP clients
- xmlrpc — XMLRPC server and client modules
- xmlrpc.client — XML-RPC client access
- xmlrpc.server — Basic XML-RPC servers
- ipaddress — IPv4/IPv6 manipulation library

23. Multimedia Services
    - audioop — Manipulate raw audio data
    - aifc — Read and write AIFF and AIFC files
    - sunau — Read and write Sun AU files
    - wave — Read and write WAV files
    - chunk — Read IFF chunked data
    - colorsys — Conversions between color systems
    - imghdr — Determine the type of an image
    - sndhdr — Determine type of sound file
    - ossaudiodev — Access to OSS-compatible audio devices
24. Internationalization
    - gettext — Multilingual internationalization services
    - locale — Internationalization services
25. Program Frameworks
    - turtle — Turtle graphics
    - cmd — Support for line-oriented command interpreters
    - shlex — Simple lexical analysis
26. Graphical User Interfaces with Tk
    - tkinter — Python interface to Tcl/Tk

- tkinter.ttk — Tk themed widgets
- tkinter.tix — Extension widgets for Tk
- tkinter.scrolledtext — Scrolled Text Widget
- IDLE
- Other Graphical User Interface Packages

27. Development Tools
    - typing — Support for type hints
    - pydoc — Documentation generator and online help system
    - doctest — Test interactive Python examples
    - unittest — Unit testing framework
    - unittest.mock — mock object library
    - unittest.mock — getting started
    - 2to3 - Automated Python 2 to 3 code translation
    - test — Regression tests package for Python
    - test.support — Utilities for the Python test suite
    - test.support.script_helper — Utilities for the Python execution tests
28. Debugging and Profiling
    - Audit events table
    - bdb — Debugger framework
    - faulthandler — Dump the Python traceback
    - pdb — The Python Debugger
    - The Python Profilers
    - timeit — Measure execution time of small code snippets
    - trace — Trace or track Python statement execution
    - tracemalloc — Trace memory allocations
29. Software Packaging and Distribution
    - distutils — Building and installing Python modules
    - ensurepip — Bootstrapping the pip installer
    - venv — Creation of virtual environments
    - zipapp — Manage executable Python zip archives
30. Python Runtime Services
    - sys — System-specific parameters and functions
    - sysconfig — Provide access to Python's configuration information
    - builtins — Built-in objects
    - __main__ — Top-level script environment
    - warnings — Warning control
    - dataclasses — Data Classes
    - contextlib — Utilities for with-statement contexts
    - abc — Abstract Base Classes
    - atexit — Exit handlers
    - traceback — Print or retrieve a stack traceback
    - __future__ — Future statement definitions
    - gc — Garbage Collector interface
    - inspect — Inspect live objects
    - site — Site-specific configuration hook
31. Custom Python Interpreters

- code — Interpreter base classes
- codeop — Compile Python code

32. Importing Modules
    - zipimport — Import modules from Zip archives
    - pkgutil — Package extension utility
    - modulefinder — Find modules used by a script
    - runpy — Locating and executing Python modules
    - importlib — The implementation of import
    - Using importlib.metadata
33. Python Language Services
    - parser — Access Python parse trees
    - ast — Abstract Syntax Trees
    - symtable — Access to the compiler's symbol tables
    - symbol — Constants used with Python parse trees
    - token — Constants used with Python parse trees
    - keyword — Testing for Python keywords
    - tokenize — Tokenizer for Python source
    - tabnanny — Detection of ambiguous indentation
    - pyclbr — Python module browser support
    - py_compile — Compile Python source files
    - compileall — Byte-compile Python libraries
    - dis — Disassembler for Python bytecode
    - pickletools — Tools for pickle developers
34. Miscellaneous Services
    - formatter — Generic output formatting
35. MS Windows Specific Services
    - msilib — Read and write Microsoft Installer files
    - msvcrt — Useful routines from the MS VC++ runtime
    - winreg — Windows registry access
    - winsound — Sound-playing interface for Windows
36. Unix Specific Services
    - posix — The most common POSIX system calls
    - pwd — The password database
    - spwd — The shadow password database
    - grp — The group database
    - crypt — Function to check Unix passwords
    - termios — POSIX style tty control
    - tty — Terminal control functions
    - pty — Pseudo-terminal utilities
    - fcntl — The fcntl and ioctl system calls
    - pipes — Interface to shell pipelines
    - resource — Resource usage information
    - nis — Interface to Sun's NIS (Yellow Pages)
    - syslog — Unix syslog library routines
37. Superseded Modules
    - optparse — Parser for command line options

- imp — Access the import internals

38. Undocumented Modules
    - Platform specific modules

Clarify code using commenting techniques

When working with any programming language, you'll be required to include and add comments to mention how a particular code works or operates or sometimes to explain the simple purpose to write the code. Comments are also included to explain what other parts of the code are all about and lets certain developers know what they were up to when they wrote the code. Use of comments is a good practice, and good developers use them quite often. Without it, things could get complicated, incredibly quick for the developers and others.

Writing comments in Python programming language

There are two options to annotate the code in Python. The first option is to include comments which explain what a code section, segment-or snippet-does. The second option takes advantage of multi-line comments or paragraphs that act as documentation to interpret the code for others.

Consider of the first type as a personal comment to your own knowledge sake, and the second type as a reference to everyone else. Nonetheless, there is no right or wrong way to include a comment. You should do whatever you feel comfortable with, but you must consider the purpose of annotating the code, is it for you or others?

Single-line comments are generated simply by beginning a line with the hash (#) character and are terminated automatically by the end of the line by pressing the enter key. Unlike other programming languages, Python doesn't support blocks out of the box for multi-line comment.

A commonly used way to include a comment on multiple lines of code in Python programming language is by using consecutive # comments on a single line.

Let's consider an example now:

```
#This is used to write comment in Python
```

Comments that span multiple lines – used to explain things in more detail – are created by adding a delimiter (“””) on each end of the comment.

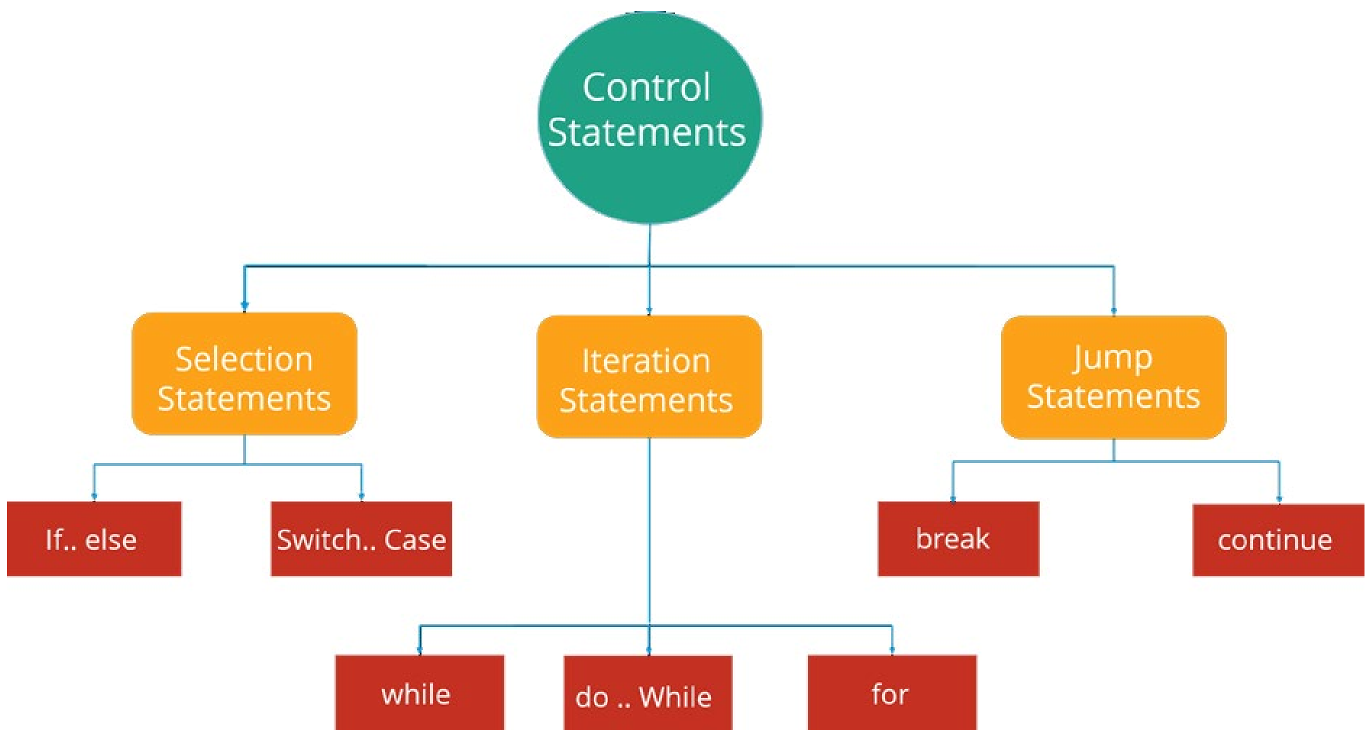
```
"""
This would be a multiline comment
in Python that spans several lines and
describes your code, your day, or anything you want it to
"""
```

Note: Triple or double quote comments (""") are used for special purposes such as documentation in the Python language for other developers to read and review the code. Therefore, these commenting techniques should not be used in the Python language.

Control structures

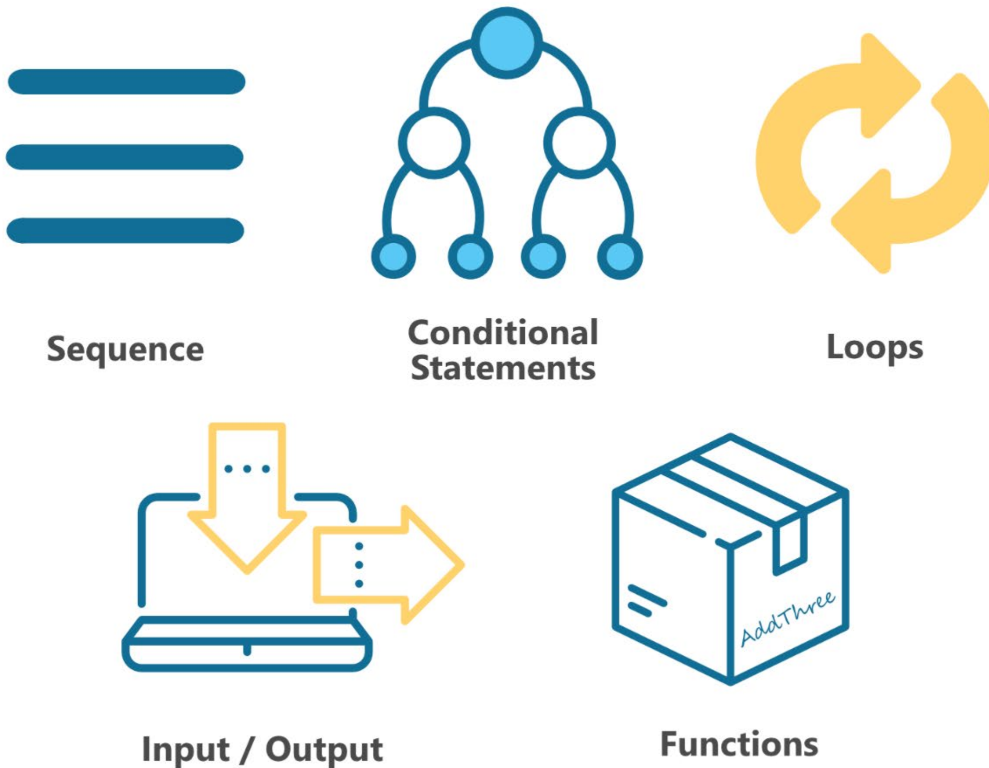
A programming language use control structure as a syntactic method to describe the flow control. A sequence of statements would be executed according to whether the statement or condition is true or false. This implies the program opts for two or more alternative paths to select from. Which is why in computer programming, it is the basic decision-making process; flow control decides how a machine will respond when given certain conditions and parameters will be executed.

Usually in a "control flow," the computer executes the instructions one by one in the sequence in which they appear. This concept is known as sequence accomplishment or accomplishment of sequence. The statement which will be executed next in computer programming is not usually placed in the next line or section. The concept is known as a transfer of program control in the programming language.



Apply language syntax in sequence, selection and iteration constructs

There are five elements we consider at this level:



Sequential

When the statements execute one after another in order without the need of anyone to do anything is known as sequential execution.

Selection

When the statements execute based on some conditions and decisions based on selecting two or more alternative paths or options.

- if
- if...else
- switch

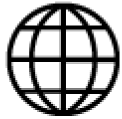
Repetition

When the statements are executed for repetition or looping, i.e. repeating a section of code multiple times in a row.

- while loop

- do..while loop
- for loop

Computer programming can use these control structures as a combination or mix. A sequence may contain several loops; a loop may contain a nested loop within it, or each of the two branches of a conditional may contain loop sequences and more conditionals. The control structures and statements in Python language can be learned from the link below.



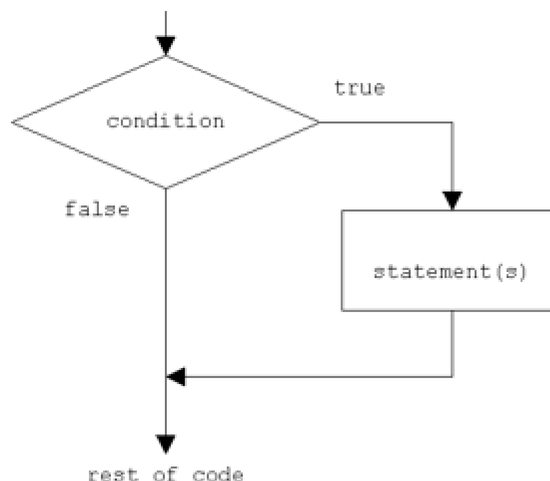
Python – Control Statements *(http://net-informations.com/python/flow/default.htm)*

The conditional statements

Decision making is one of computer programming's most important concepts. This requires the developer to define one or more conditions to be checked or tested by the program, along with a statement or statements to be executed if the condition is determined to be true or valid and, if the condition is determined to be invalid or false, other statements to be executed. Python programming language offers decision-making statements of the following types.

- if statements
- if....else statements
- if..elif..else statements
- nested if statements
- not operator in if statement
- and operator in if statement
- in operator in, if statement

How if statements flow in the Python programming language:



In Python, 'if' statement evaluates the test expression inside the parenthesis. If the test expression is evaluated to true (nonzero), statements inside the body of 'if' they are executed. If the test expression is evaluated to false (0), statements inside the body of 'if' are skipped.

Example:

```
x=20
y=10
if x > y :
  print(" X is bigger ")
```

Output

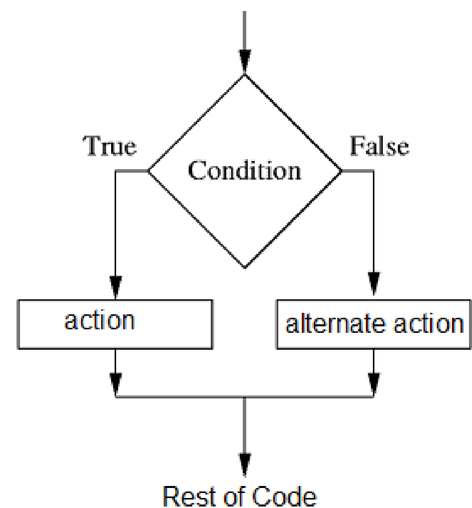
```
X is bigger
```

In this program, we have two variables x and y. x is assigned as the value 20, and y is 10. In the next line, the if statement evaluates the expression (x>y) is true or false. In this case, the x > y is true because x=20 and y=10, then the control goes to the body of if block and print the message "X is bigger". If the condition is false, then the control goes outside the if block.

Python if..else statements

The else statement is to specify a block of code to be executed if the condition in the if the statement is false. Thus, the else clause ensures that a sequence of statements is executed.

```
if expression:
  statements
else:
  statements
```



Example:

```
x=10
y=20
if x > y :
  print(" X is bigger ")
else :
  print(" Y is bigger ")
```

Output:

```
Y is bigger
```

In the above code, the, if statement evaluates the expression, being true or false. In this case, the x > y is false, then the control goes to the body of else block, so the program will execute the code inside else block.

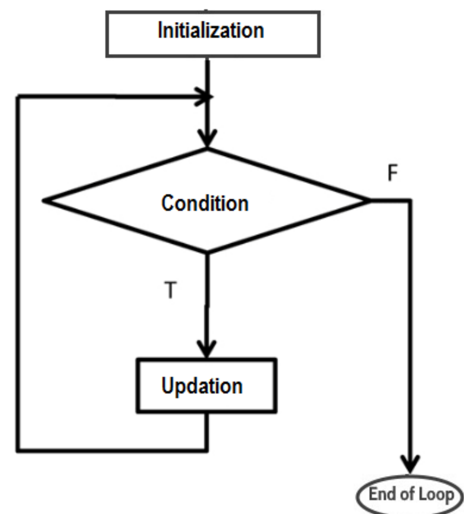
Python while loop Statements

Loops are one of the most important features in computer programming languages. As the name suggests is the process that gets repeated again and again. It offers a quick and easy way to do something repeated until a certain condition is reached. Every loop has 3 parts:

Initialisation

Condition

Updating



Python while loop

Syntax

```
while (condition) :
  statement(s)
```

In Python, while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. That means while the loop tells the computer to do something as long as the condition is met. It consists of condition/expression and a block of code. The condition/expression is evaluated, and if the condition/expression is true, the code within the block is executed. This repeats until the condition/expression becomes false.

```
initialization;
while(condition)
{
  //Code block to execute something
}
```

For example, if the value of a variable x is initialized as 0 and the condition is set x < =5, then the condition will be held true. But if the condition is set to x>=5 the condition will become false. After checking the condition in a 'while' clause, if it holds true, the body of the loop is executed. While executing the body of the loop, it can update the statement inside while loop. After updating, the condition is checked again. This process is repeated so long as the condition is true, and once the condition becomes false, the program breaks out of the loop.

Example:

```
x=0
while(x <=5):
  print(x)
  x+=1
```

Output

0

1

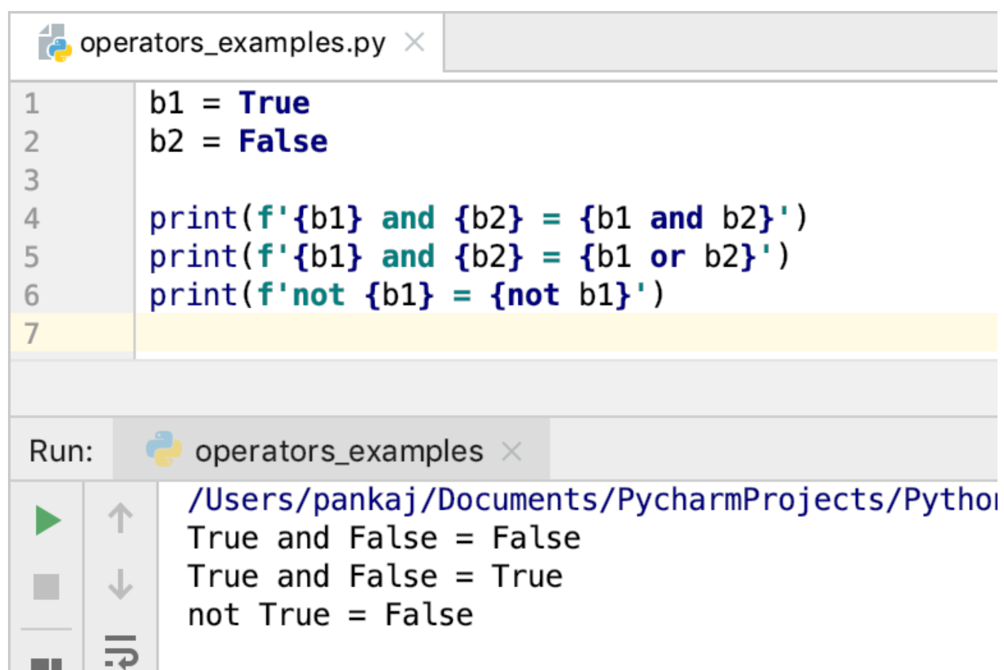
2

3

4

5

Create expressions in selection and iteration constructs using logical operators



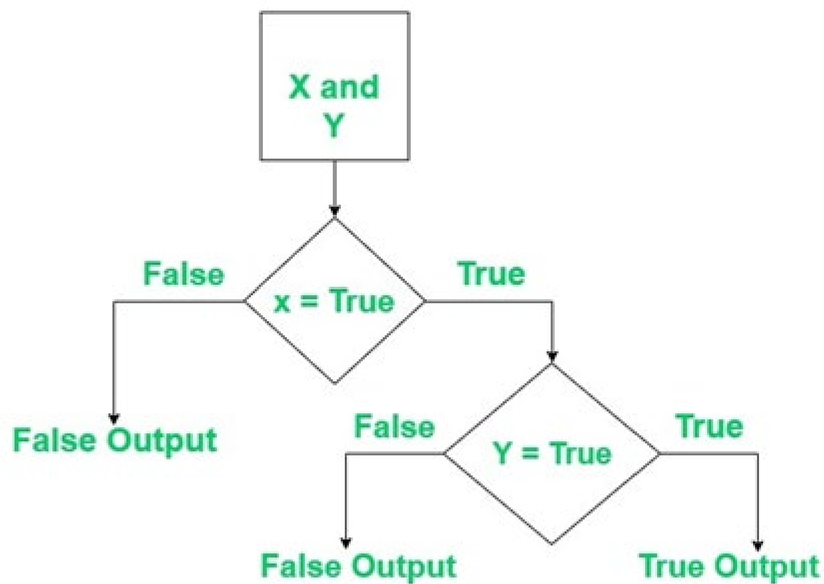
Logical operators

In Python, Logical operators are used on conditional statements (either True or False). They perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

Logical AND operator

Logical operator returns True if both the operands are True else it returns False.



Example:

```
# Python program to demonstrate
# logical and operator

a = 10
b = 10
c = -10

if a > 0 and b > 0:
        print("The numbers are greater than 0")

if a > 0 and b > 0 and c > 0:
        print("The numbers are greater than 0")
```

```
else:
    print("At least one number is not greater than 0")
```

Output

```
The numbers are greater than 0
At least one number is not greater than 0
```

Example:

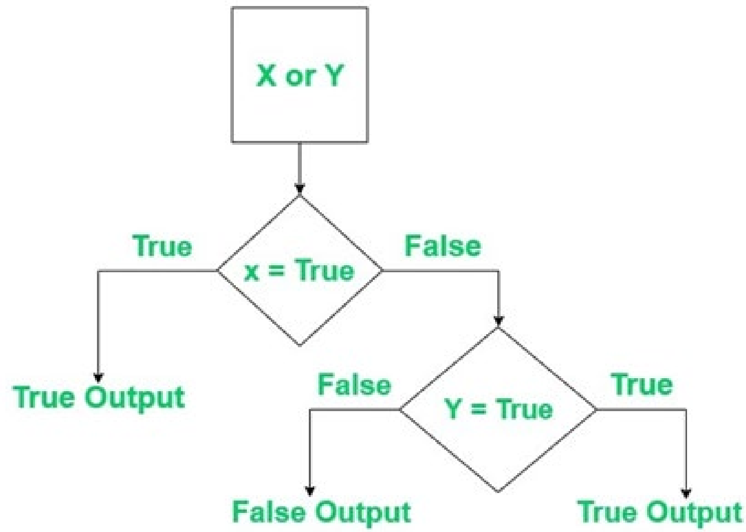
```
# Python program to demonstrate
# logical and operator

a = 10
b = 12
c = 0

if a and b and c:
    print("All the numbers have boolean value as True")
else:
    print("At least one number has boolean value as False")
```

Output

```
At least one number has boolean value as False
```

Logical or operator

Logical or operator returns True if either of the operands is True.



Example:

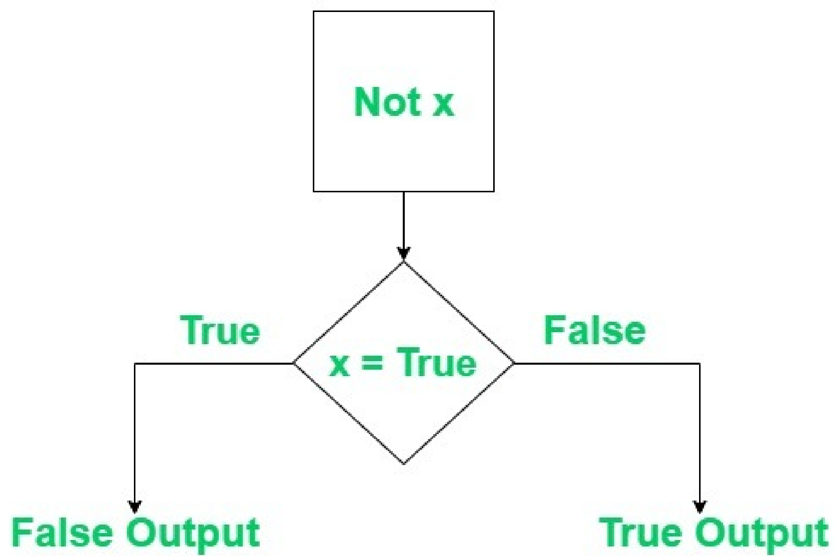
```
# Python program to demonstrate
# logical or operator


a = 10
b = -10
c = 0


if a > 0 or b > 0:
	print("Either of the number is greater than 0")
else:
	print("No number is greater than 0")


if b > 0 or c > 0:
	print("Either of the number is greater than 0")
else:
	print("No number is greater than 0")
```

Output:

```
Either of the number is greater than 0
No number is greater than 0
```

Logical not operator

Logical not operator work with the single boolean value. If the boolean value is True, it returns False and vice-versa.



Example:

```
# Python program to demonstrate
# logical not operator

a = 10

if not a:
    print("Boolean value of a is True")

if not (a%3 == 0 or a%5 == 0):
    print("10 is not divisible by either 3 or 5")
else:
    print("10 is divisible by either 3 or 5")
```

Output:

    10 is divisible by either 3 or 5

Code Using Standard Algorithms

When you write a program, you need to tell the computer every small detail of what to do. You need to get everything right since the computer will blindly follow your program exactly as it is written. A program is an expression of an idea. A programmer starts with a general idea of a task for the computer to perform. Presumably, the programmer has some idea of how to perform the task at least in general outline.

Develop algorithms using sequence, selection and iteration constructs

An algorithm is a step by step process that describes how to solve a problem in a way that always gives a correct answer. When there are multiple algorithms for a particular problem (and there often are!), the best algorithm is typically the one that solves it the fastest.

As computer programmers, we are constantly using algorithms, whether it's an existing algorithm for a common problem, like sorting an array, or if it's a completely new algorithm unique to our program. By understanding algorithms, we can make better decisions about which existing algorithms to use and learn how to make new algorithms that are correct and efficient.

An algorithm is made up of three basic building blocks: sequencing, selection, and iteration.

Sequencing: An algorithm is a step-by-step process, and the order of those steps are crucial to ensuring the correctness of an algorithm.

Selection: Algorithms can use selection to determine a different set of steps to execute based on a Boolean expression.

Iteration: Algorithms often use repetition to execute steps a certain number of times or until a certain condition is met.

It is a remarkable result of computer science that these basic constructs can be used to implement any programming problem!

The basic constructs of sequence, selection and iteration can be embedded within each other, and this is made clear by using indenting. Nested constructs should be indented from their surrounding constructs.

"The machine learning algorithm wants to know if we'd like a dozen wireless mice to feed the Python book we just bought."

Nesting IF constructs

The IF-THEN-ELSE construct can be nested inside another IF-THEN-ELSE construct.

Pseudocode	Flowchart

IF cat whining for food THEN
  IF cat bowl empty THEN
    Feed cat
  ELSE
    Point cat at food
  ENDIF
ENDIF
	START
	Cat whining ?
	No
	Yes
	Cat bowl empty ?
	Yes
	No
	Feed cat
	Point cat at bow
	END

Here we first test to see if the cat is whining for food. If it isn't, we simply end the testing. If the cat is whining for food, then we will execute the other IF-THEN-ELSE construct:

IF food bowl empty THEN

  Feed the cat

ELSE

  Point cat at food bowl

ENDIF

Pseudocode—nested constructs

Pseudocode	Flowchart

ET maxTemp = -100
REPEAT
READ temp
IF temp > maxTemp THEN
SET maxTemp = temp
ENDIF
READ time
UNTIL time == MIDNIGHT
PRINT maxTemp
	START
	SET maxTemp = -100
	READ temp
	temp > maxTemp
	Yes
	SET maxTemp = temp
	No
	READ time
	time > MIDNIGHT
	No
	Yes
	DISPLAY maxTemp
	END

In the above example, the IF construct is nested within the REPEAT construct, and therefore is indented.

Designing algorithms

A simple method for designing a simple program could be:

1. Understand the problem or requirements.
2. Identify the inputs and the outputs and decide what variables are needed.
3. Determine the calculations/computations that are required to produce the output.
4. To construct the algorithm, start by assigning input values to the variables.
5. Then perform the computations.
6. Generate the output.
7. When there are processes to be repeated, add loops.
8. When decisions are to be made, use selection.

Create and use data structures

ARRAYS

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the index number.

Example:

Get the value of the first array item:

```
x = cars[0]
```

Example:

Modify the value of the first array item:

```
cars[0] = "Toyota"
```

The Length of an Array

Use the 'len()' method to return the length of an array (the number of elements in an array).

Example:

Return the number of elements in the 'cars' array:

```
x = len(cars)
```

Looping Array Elements

You can use the 'for in' loop to loop through all the elements of an array.

Example:

Print each item in the 'cars' array:

```
for x in cars:
  print(x)
```

Adding Array Elements

You can use the 'append()' method to add an element to an array.

Example:

Add one more element to the 'cars' array:

```
cars.append("Honda")
```

Removing Array Elements

You can use the 'pop()' method to remove an element from the array.

Example:

Delete the second element of the 'cars' array:

```
cars.pop(1)
```

You can also use the 'remove()' method to remove an element from the array.

Example:

Delete the element that has the value "Volvo":

```
cars.remove("Volvo")
```

Array Methods

Python has a set of built-in methods that you can use on lists/arrays. Click on the method name below for more information.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Code standard sequential access algorithms used in reading and writing text files

Sequences are one of the principal built-in data types besides numerics, mappings, files, instances and exceptions. Python provides for six sequences (or sequential) data types:

- strings
- byte sequences
- byte arrays
- lists
- tuples
- range objects

Strings, lists, tuples, bytes and range objects may look like utterly different things, but they still have some underlying concepts in common:

- The items or elements of strings, lists and tuples are ordered in a defined sequence.
- The elements can be accessed via indices.

```
text = "Lists and Strings can be accessed via indices!"
print(text[0], text[10], text[-1])
L S !
```

Accessing lists:

```
lst = ["Vienna", "London", "Paris", "Berlin", "Zurich", "Hamburg"]
print(lst[0])
print(lst[2])
print(lst[-1])
Vienna
Paris
Hamburg
```

Unlike other programming languages Python uses the same syntax and function names to work on sequential data types. For example, the length of a string, a list, and a tuple can be determined with a function called len():

```
countries = ["Germany","Switzerland","Austria","France","Belgium", "Netherlands", "Engl
and"]
```

```
len(countries)
```

Output::

```
7
```

```
fib = [1,1,2,3,5,8,13,21,34,55]
len(fib)
```

Output::

```
10
```

Apply string manipulation

Strings in Python

Strings are one of the most basic data types in Python, used to represent textual data. Almost every application involves working with strings, and Python's str class provides a variety of methods to make string manipulation easy.

Basic String Operations

Define a String

Strings are denoted with either single or double quotes:

```
string_1  = "Example string #1"
string_2 = 'Example string #2'
```

Both methods are equivalent. If a string is delimited with double quotes, any double quotation marks within the string will need to be escaped with a backslash (\):

```
“My teacher said \”Don’t forget your homework.\””
```

Similarly, in single-quoted strings, you will need to escape any apostrophes or single-quoted expressions:

```
'This is Linode\'s documentation site.'
```

Subset Strings

Python does not have a Character data type. To access individual characters within a string, use bracket notation. Like lists, Python strings are zero-indexed, so the first character of a string can be accessed with [0]:

```
string_3 = "This is a string."
first_letter = string_3[0]
```

To access a range of letters from a larger string, use slicing:

```
string_3[0:4]
```

This will return all characters starting from the number before the colon (0, or the first character) up to but not including the index after the colon (4). In this case, the first four letters of the string are returned:

```
'This'
```

String Operators

The + and * operators are overridden for the string class, making it possible to add and multiply strings. Strings in Python are immutable. They cannot be modified after being created.

Using the add operator to combine strings is called concatenation. The strings for the first and last name remain unchanged. Concatenating the two strings returns a new string.

```
first_name = "Abraham"
last_name = " Lincoln"
first_name + last_name

Abraham Lincoln
```

Multiplication can be used to generate multiple copies of strings:

```
"a" * 10

'aaaaaaaaaa'
```

String Methods

Many basic string manipulation tasks can be handled with built-in methods. For example, to convert a string to uppercase letters, use upper:

```
'example string'.upper()
```

EXAMPLE STRING

To remove extra whitespace from the beginning or end of a string, use strip:

'example '.strip()

'example'

Strings can be split into a list of substrings with a split. By default, Python will use a blank space as a delimiter, which is useful for splitting a sentence into individual words:

'This string has five words'.split()

['This', 'string', 'has', 'five', 'words']

Specify a different delimiter by passing the character(s) as an argument to split:

'one,two,three,four,five'.split(',')

['one', 'two', 'three', 'four', 'five']

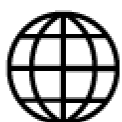
The inverse operation of a split is 'join', which will combine a list of strings into a single string. The join method must be called on a string that will be used to separate the list entries in the final string:

' '.join(['This', 'string', 'has', 'five', 'words'])

'This string has five words'

','.join(['one', 'two', 'three', 'four', 'five'])

'one,two,three,four,five'



String Methods - Python's official documentation.
(https://docs.python.org/3/library/stdtypes.html#string-methods)

String Formatting

Often strings need to be built on the fly, based on the state of the application. For example, you may want to customize an error message with information about the values that caused the error. There are several ways to accomplish this in Python; this section will review two of the commonly used methods in Python 3.

str.format()

Prior to Python 3.6, the 'str.format()' method was arguably the easiest and most convenient way to format strings. Each string object has access to the 'format' method, which allows substituting values into the string at designated locations:

```
name, age = "Alice", 26
string_template = 'My name is {0} and I am {1} years old.'
string_template.format(name, age)
```

The 'format' method is called on the 'string_template' object. The 'format' takes as arguments a comma-separated list of variables to insert into the string calling the method. The variables will be substituted into the bracketed portions of the string. The first argument ('name' is argument zero since Python lists are zero-indexed) is substituted into the string in place of '{0}', and 'age' is substituted for '{1}'. Any number of substitutions can be made in this way.

If numbers between the brackets are omitted, Python will substitute the variables in the order in which they are passed to 'format':

```
snack, hobby = "avocado", "tail recursion"
string_template = 'My name is {} and I am {} years old. I enjoy {} and {}.'
string_template.format(name, age, snack, hobby)
```

This is equivalent to:

```
'My name is {0} and I am {1} years old. I enjoy {2} and {3}.'
```

This syntax is often shortened by combining the string declaration and 'str.format' method call into a single statement:

```
'My name is {} and I am {} years old. I enjoy {} and {}.'.format(name, age, snack, hobby)

'My name is Alice and I am 26 years old. I enjoy avocado and tail recursion.'
```

Finally, recall that a variable is just one type of expression in Python, and other expressions can usually be used in place of a variable. This is true when formatting strings, where arbitrary expressions can be passed to 'str.format':

```
fahrenheit = 54
```

'The temperature is {} degrees F ({} degrees C).'.format(fahrenheit, int((fahrenheit - 32) * (5/9.0)))

f-strings

Python 3.6 introduced a simpler way to format strings: formatted string literals usually referred to as **f-strings**.

ram, region = 4, 'us-east'

f'This Linode has {ram}GB of RAM, and is located in the {region} region.'

'This Linode has 4GB of RAM, and is located in the us-east region.'

The 'f' at the beginning of the above string designates it as an f-string. The syntax is similar to the 'str.format()' method. Variable names can be placed directly inside the string itself enclosed in brackets rather than in a function call following the string. This makes f-strings more compact and readable.

Any Python expression can be placed inside the brackets in an f-string, giving them even more flexibility:

orders = [14.99,19.99,10]

f'You have {len(orders)} items in your cart, for a total cost of ${sum(orders)}.'

'You have 3 items in your cart, for a total cost of $44.98.'

Test Code

What Is Testing?

Testing is a process of exploring the system to find defects present in the software, and to define what will happen once these defects occur. This process is performed in the testing phase by the testing team, and after this phase, they will report to the developer team to debug.

Objectives of software testing:

- Ensures the quality of product
- Defect prevention and detection
- Verify and validate the user requirement
- Ready to integration and revise the component
- Focus on the accurate and reliable result
- Work should be performed under predefined process and SRS
- Discuss on generating test cases test cases
- Provide information to decide for the next phase
- Gain confidence in work
- Evaluates the capabilities of a system and system performance

Principles of software testing:

- It is an ongoing process to achieve a quality product
- Defect discovered, Analysis and correct
- Trying to meet user satisfaction
- Ensure product and system application
- Evaluation of user requirement
- Covers all test coverage
- Process for early testing to discover a defect

Types of Software Testing

Alpha Testing

It is the most common type of testing used in the Software industry. The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user. Alpha testing is carried out at the end of the software development phase but before

the Beta Testing. Still, minor design changes may be made as a result of such testing. Alpha testing is conducted at the developer's site. In-house virtual user environment can be created for this type of testing.

Acceptance Testing

An acceptance test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end-user. Client accepts the software only when all the features and functionalities work as expected. It is the last phase of the testing, after which the software goes into production. This is also called as User Acceptance Testing (UAT)

Ad-hoc Testing

The name itself suggests that this testing is performed on an ad-hoc basis, i.e. with no reference to test case and also without any plan or documentation in place for such type of testing. The objective of this testing is to find the defects and break the application by executing any flow of the application or any random functionality.

Ad-hoc testing is an informal way of finding defects and can be performed by anyone in the project. It is difficult to identify defects without a test case. Still, sometimes it is possible that defects found during ad-hoc testing might not have been identified using existing test cases.

Beta Testing

Beta Testing is a formal type of software testing which is carried out by the customer. It is performed in Real Environment before releasing the product to the market for the actual end-users. Beta testing is carried out to ensure that there are no major failures in the software or product, and it satisfies the business requirements from an end-user perspective. Beta testing is successful when the customer accepts the software.

Usually, this testing is typically done by end-users or others. It is the final testing done before releasing an application for commercial purpose. Usually, the Beta version of the software or product released is limited to a certain number of users in a specific area. So end-user uses the software and shares the feedback to the company. Company then takes necessary action before releasing the software to the worldwide.

System Integration Testing

known as SIT (in short) is a type of testing conducted by software testing team. As the name suggests, the focus of System integration testing is to test for errors related to integration among different applications, services, third party vendor applications etc.; As part of SIT, end-to-end scenarios are tested that would require software to interact (send or receive data) with other upstream or downstream applications, services, third party application calls etc.,

Unit testing

is a type of testing that is performed by software developers. Unit testing follows the white box testing approach where a developer will test units of source code like statements,

branches, functions, methods, interface in OOP (object-oriented programming). Unit testing usually involves in developing stubs and drivers. Unit tests are ideal candidates for automation. Automated tests can run as Unit regression tests on new builds or new versions of the software. There are many useful unit testing frames works like Junit, Nunit, etc., available that can make unit testing more effective.

User Acceptance Testing (UAT)

User Acceptance testing is a must for any project; it is performed by clients/end users of the software. User Acceptance testing allows SMEs (Subject matter experts) from client to test the software with their actual business or real-world scenarios and to check if the software meets their business requirements.

White Box Testing

White box testing is also known as clear box testing, transparent box testing and glass box testing. White box testing is a software testing approach, which intends to test software with knowledge of the internal working of the software. White box testing approach is used in Unit testing, which is usually performed by software developers. White box testing intends to execute code and test statements, branches, path, decisions and data flow within the program being tested.

Black Box Testing

Black box testing is a software testing method where testers are not required to know coding or internal structure of the software. Black box testing method relies on testing software with various inputs and validating results against expected output.

Performance Testing

This is a type of software testing and part of performance engineering that is performed to check some of the quality attributes of software like Stability, reliability, availability. The performance engineering team carries out performance testing. Unlike Functional testing, Performance testing is done to check non-functional requirements. Performance testing checks how well the software works in anticipated and peak workloads. There are different variations or subtypes of performance like load testing, stress testing, volume testing, soak testing and configuration testing.

What Is Debugging?

The word "bug" suggests something that wandered into a program. Better terminology would be that there are faults, which are the result of human errors in software systems, and failures, which are violations of requirements.

Debugging is the process of discovering and fixing faults. Testing is the "discovery" part but fixing can be more complicated. Debugging can be a task that takes even more time than an original implementation itself! So you would do well to make it easy to debug your programs from the start. Write good specifications for each function

Requirements for Debugging

To effectively debug code, you need two capabilities. First, you need to be able to efficiently call on the services provided by the module. Then you need to be able to get information back about results of the calls, changes in the internal state of the module, error conditions, and what the module was doing when an error occurred.

To effectively debug a module, it is necessary to have some method for calling upon the services provided by the module. There are two common methods for doing this.

Hardwired drivers: A hardwired driver is the main program module that contains a fixed sequence of calls to the services provided by the module that is being tested. The sequence of calls can be modified by rewriting the driver code and recompiling it. For testing modules whose behaviour is determined by a very small number of cases, hardwired drivers offer the advantage of being easy to construct. If there are too many cases, though, they have the shortcoming that a considerable effort is involved in modifying the sequence of calls.

Command interpreters: A command interpreter drives the module under test by reading input and interpreting it as commands to execute calls to module services. Command interpreters can be designed so that the commands can either be entered interactively or read from a file. Interactive command interpretation is often of great value in early stages of debugging, whereas batch mode usually is better for later stages of debugging and final testing. The primary disadvantage of command interpreters is the complexity of writing one, including the possibility that a lot of time can be spent debugging the interpreter code. This is mitigated by the fact that most of the difficult code is reusable and can be easily adapted for testing different kinds of modules. For almost all data structure modules, the flexibility offered by command interpreters makes them a preferred choice.

Principles of Debugging

Report error conditions immediately - Much debugging time is spent zeroing in on the cause of errors. The earlier an error is detected, the easier it is to find the cause. If an incorrect module state is detected as soon as it arises then the cause can often be determined with minimal effort. If it is not detected until the symptoms appear in the client interface, then it may be difficult to narrow down the list of possible causes.

Maximize useful information and ease of interpretation: It is obvious that maximizing useful information is desirable, and that it should be easy to interpret. Ease of interpretation is important in data structures. Some module errors cannot easily be detected by adding code checks because they depend on the entire structure. Thus, it is important to be able to display the structure in a form that can be easily scanned for correctness.

Minimize useless and distracting information: Too much information can be as much of a handicap as too little. If you need to work with a printout that shows entry and exit from every procedure in a module, then you will find it very difficult to find the first place where something went wrong. Ideally, module execution state reports should be issued only when an error has occurred. As a general rule, debugging information that says "the problem is here" should be preferred in favour of reports that say "the problem is not here".

Avoid complex one-use testing code: One reason why it is counterproductive to add module correctness checks for errors that involve the entire structure is that the code to do so can be quite complex. It is very discouraging to spend several hours debugging a problem, only to find that the error was in the debugging code, not the module under test. Complex testing code is only practical if the difficult parts of the code are reusable.

Use debugging techniques to detect and correct errors

Debugging is the routine process of locating and removing bugs, errors or abnormalities from programs. It's a must-have skill for any Python developer because it helps to find a subtle bug that is not visible during code reviews or that only happens when a specific condition occurs. The Python IDE provides many debugging tools and views grouped in the **Debug Perspective** to help you as a developer debug effectively and efficiently.

When you run your application for the very first time, PyCharm automatically creates the temporary Run/Debug configuration. You can modify it to specify or alter the default parameters and save it as a permanent Run/Debug configuration.

Prerequisites

Before you start, ensure that Python and related frameworks are installed.

Use this dialog to create a run/debug configuration for **Python scripts**.

Configuration tab

Item	Description
Script path/Module name	Click the list to select a type of target to run. Then, in the corresponding field, specify the path to the Python script or the module name to be executed.
Parameters	In this field, specify parameters to be passed to the Python script. When specifying the script parameters, follow these rules: • Use spaces to separate individual script parameters. • Script parameters containing spaces should be delimited with double quotes, for example, some" "param or "some param". • If script parameter includes double quotes, escape the double quotes with backslashes, for example: -s"main.snap_source_dirs=[\"pcomponents/src/main/python\"]" -s"http.cc_port=8189" -s"backdoor.port=9189" -s"main.metadata={\"location\": \"B\", \"language\": \"python\", \"platform\": \"unix\"}"

Create and conduct simple tests and confirm code meets design specification

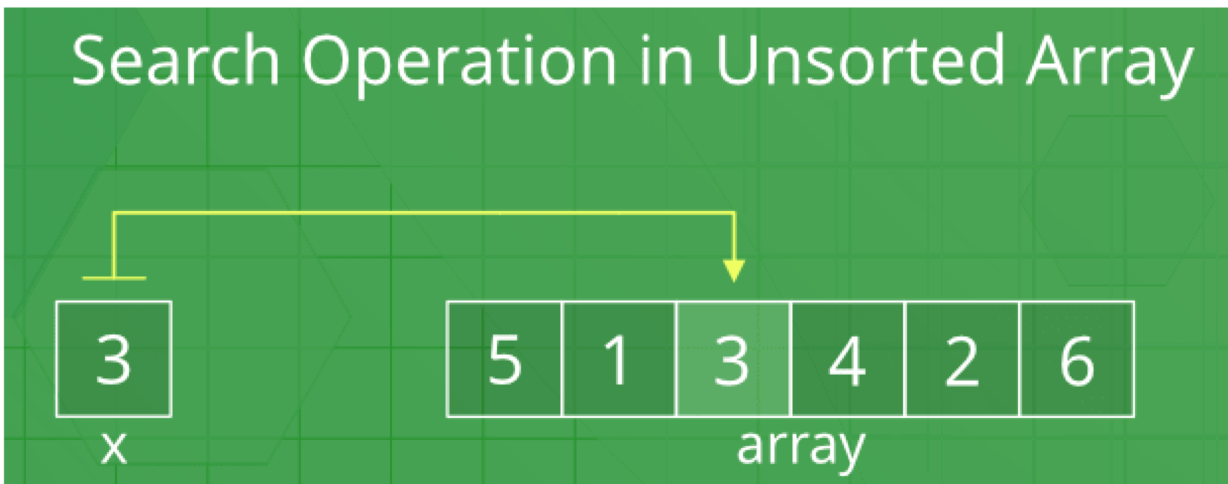
What is design specification?

A **design specification** is a detailed document providing information about a designed product or process. For example, the design specification must include all necessary drawings, dimensions, environmental factors, ergonomic factors, aesthetic factors, maintenance that will be needed, etc. Design Specifications may include:

- Specific inputs, including data types, to be entered into the system
- Calculations/code used to accomplish defined requirements
- Outputs generated from the system
- Explaining technical measures to ensure system security
- Identify how the system meets applicable regulatory requirements

Let's take an example using the search operation.

In an unsorted array, the search operation can be performed by linear traversal from the first element to the last element.




```
# Python program for searching in
# unsorted array

def findElement(arr, n, key):
    for i in range (n):
        if (arr[i] == key):
            return i
    return -1
```

```
arr = [12, 34, 10, 6, 40]
key = 40
n = len(arr)

#search operation
index = findElement(arr, n, key)
if index != -1:
	print ("element found at position: " + str(index + 1 ))
else:
	print ("element not found")

# Thanks to Aditi Sharma for contributing
# this code
```

Output:

Element Found at Position: 5

Document actions carried out and results of tests performed

You must document the tests performed and results achieved using the test case templates.

There are variety of templates that can be used to record the test activities.

Example - Python program to add two numbers

```
# This program adds two numbers

num1 = 15.5
num2 = 16.1

# Add two numbers
sum = num1 + num2

# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

Outcome:

```
The sum of 15.5 and 16.1 is 31.6
>>>
```

Example 2 - Python Program to Generate a Random Number

```
# Program to generate a random number between 0 and 100

# importing the random module
import random

print(random.randint(0,100))
```

Outcome:

```
19
>>>
```

Test Case

TEST CASE # *(from requirements document)*	x
PRODUCT BEING TESTED	x
SCENARIO	x Testing that will take place. A) X B) X C) X D) X
ACCEPTANCE CRITERIA:	

Identify any data that needs to be set up for this test.

No.	STEPS	EXPECTED RESULTS	TESTER INITIAL	PASS/ FAIL ACTUAL RESULTS
1.	sum = num1 + num2	Sum of two numbers	Sum of num 1 and num 2	Pass
2.	print(random.randint(0,100))	A random number between 0 and 100.	Any number between 0 and 100.	Pass

Create a Simple Application and Seek Feedback

Design an algorithm in response to basic program specifications

We are going to discuss Python program to check the prime number in this section.

A positive integer greater than 1 which has no other factors except 1 and the number itself is called a prime number. 2, 3, 5, 7 etc. are prime numbers as they do not have any other factors. But 6 is not prime (it is composite) since, 2 x 3 = 6.

Source Code

```
# Program to check if a number is prime or not
num = 407
# To take input from the user
#num = int(input("Enter a number: "))
# prime numbers are greater than 1
if num > 1:
  # check for factors
  for i in range(2,num):
      if (num % i) == 0:
          print(num,"is not a prime number")
          print(i,"times",num//i,"is",num)
          break
  else:
      print(num,"is a prime number")
# if input number is less than
# or equal to 1, it is not prime
else:
  print(num,"is not a prime number")
```

Output:

407 is not a prime number

11 times 37 is 407

In this program, variable num is checked if it's prime or not. Numbers less than or equal to 1 are not prime numbers. Hence, we only proceed if the 'num' is greater than 1.

We check if 'num' is exactly divisible by any number from 2 to 'num – 1'. If we find a factor in that range, the number is not prime. Else the number is prime.

We can decrease the range of numbers where we look for factors.

In the above program, our search range is from 2 to num - 1.

We could have used the range, 'range(2,num//2)' or 'range(2,math.floor(math.sqrt(num)))'. The latter range is based on a composite number having a factor less than the square root of that number. Otherwise, the number is prime.

You can change the value of variable 'num' in the above source code to check whether a number is prime or not for other integers.

Develop application to meet program specification

Using this code, we will develop an application/game to meet the program specification.

Overview

This is a Python script of the classic game "Hangman". A row of dashes represents the word to guess. If the player guesses a letter which exists in the word, the script writes it in all its correct positions. The player has 10 turns to guess the word. You can easily customize the game by changing the variables.

Hangman Script

Make sure that you understand what each line does.

```
#importing the time module
import time

#welcoming the user
name = raw_input("What is your name? ")

print "Hello, " + name, "Time to play hangman!"

print "
"

#wait for 1 second
time.sleep(1)

print "Start guessing..."
```

```
time.sleep(0.5)

#here we set the secret
word = "secret"

#creates an variable with an empty value
guesses = ''

#determine the number of turns
turns = 10

# Create a while loop

#check if the turns are more than zero
while turns > 0:

    # make a counter that starts with zero
    failed = 0

    # for every character in secret_word
    for char in word:

    # see if the character is in the players guess
        if char in guesses:

        # print then out the character
            print char,

        else:

        # if not found, print a dash
            print "_",
```

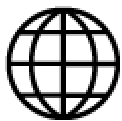
```
        # and increase the failed counter with one
            failed += 1

    # if failed is equal to zero

    # print You Won
    if failed == 0:
        print "
You won"

    # exit the script
        break

    print

    # ask the user go guess a character
    guess = raw_input("guess a character:")

    # set the players guess to guesses
    guesses += guess

    # if the guess is not found in the secret word
    if guess not in word:

     # turns counter decreases with 1 (now 9)
        turns -= 1

    # print wrong
        print "Wrong
"

    # how many turns are left
        print "You have", + turns, 'more guesses'
```

```
# if the turns are equal to zero
 if turns == 0:

 # print "You Lose"
  print "You Lose"
```

Meeting specifications

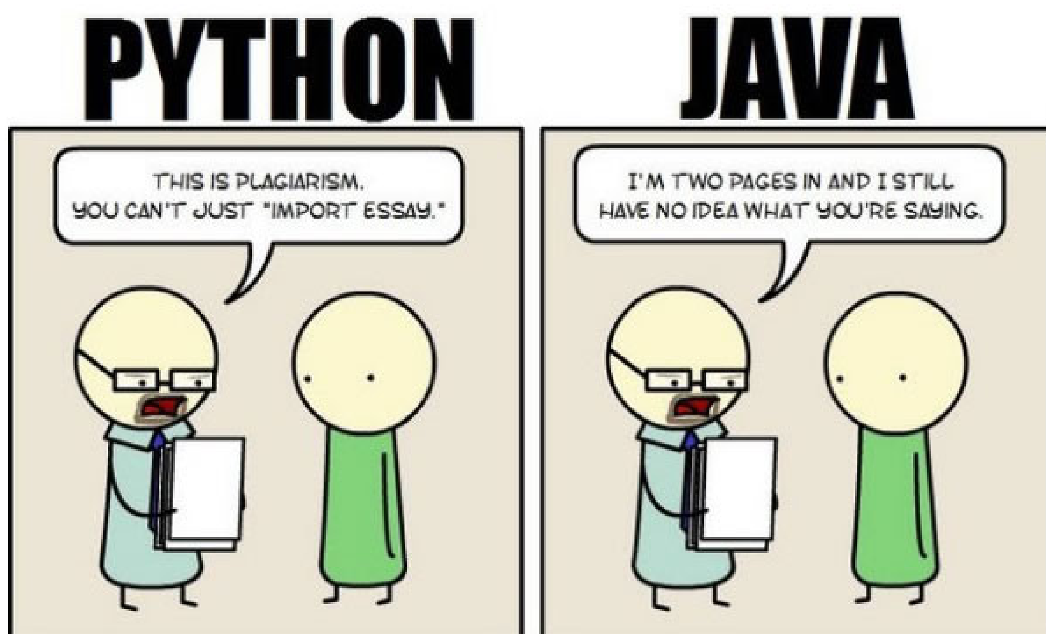
You will be required to confirm if the application meets the initial specifications.

You will find the relevant information from the following web pages:



Feedback and Review

- https://www.python.org/shell/
- https://www.onlinegdb.com/online_python_debugger
- https://amrdraz.github.io/python-debugger/
- https://repl.it/languages/python3



References

1. Codecademy, Learn PHP (2019). *Learn PHP | Codecademy*. [online] Codecademy. https://www.codecademy.com/learn/learn-php.
2. Daly, J. (2013). *A Brief History of Computer Programming Languages [#Infographic]*. https://edtechmagazine.com/higher/article/2013/04/brief-history-computer-programming-languages-infographic.
3. Guru99.com. (2019). *What is PHP? Write your first PHP Program*. https://www.guru99.com/what-is-php-first-php-program.html.
4. Ntchosting.com. (2019). *Web Hosting Services, VPS Servers and Domain Names by NTC Hosting*. https://www.ntchosting.com/encyclopedia/scripting-and-programming/php/
5. Tutorialspoint.com. (2019). *PHP Tutorial - Tutorialspoint*. https://www.tutorialspoint.com/php/index.htm.
6. W3schools.com. (2019). *PHP Tutorial*. https://www.w3schools.com/php/default.asp.
7. www.datarecoverylabs.com. (n.d.). *The History of Computer Programming Languages*. https://www.datarecoverylabs.com/company/resources/history-computer-programming-languages
8. www.mindsmapped.com. (n.d.). *You are being redirected...* https://www.mindsmapped.com/java-j2ee/java-advantages-and-disadvantages/
9. www.programiz.com. (n.d.). *Java Operators: Arithmetic, Relational, Logical and more*. https://www.programiz.com/java-programming/operators.