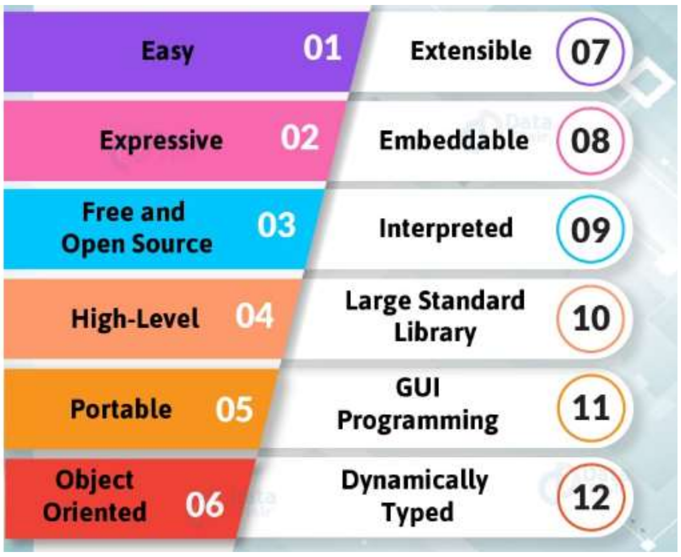
Hello Python!

Python is a very simple language, and has a very straightforward syntax. It encourages programmers to program without boilerplate (prepared) code. The simplest directive in Python is the "print" directive - it simply prints out a line.

So lets go through the video 1 – which talks more about creating your First HelloWorld program in Python.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.

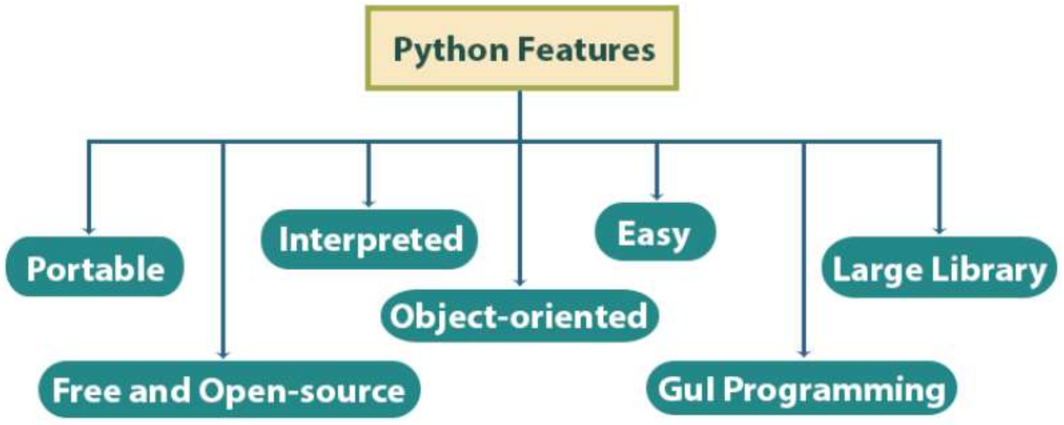


Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Important Points to Remember:

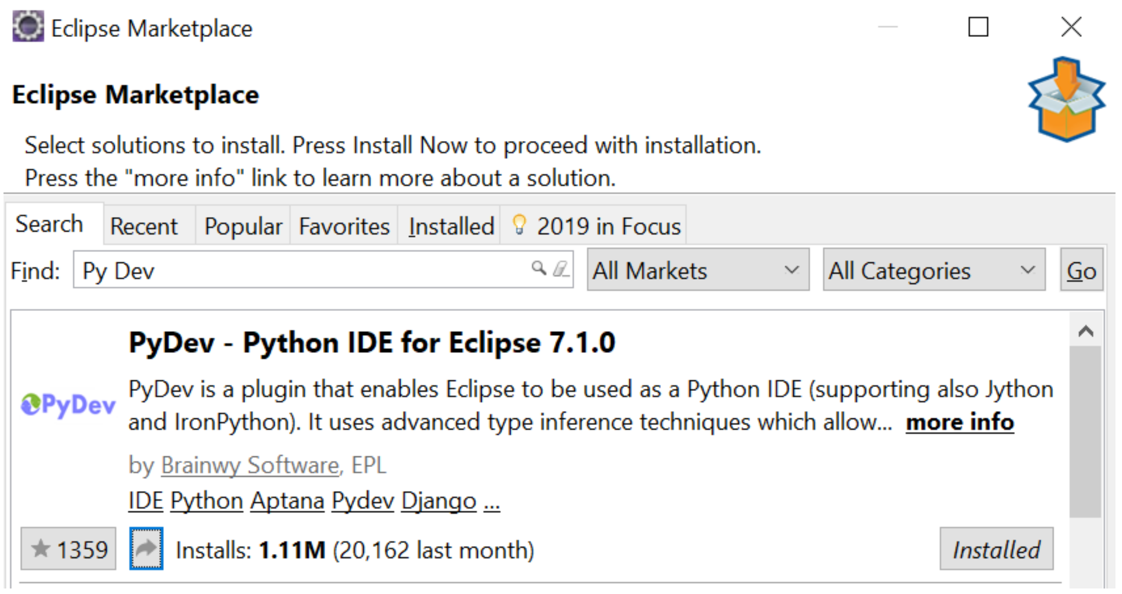
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.



RISHI (rishi.h2kinfosys@gmail.com)

First Step to Start:

1. **Download and Unzip Python** : https://www.python.org/downloads/
2. *Download and Unzip Eclipse: https://www.eclipse.org/downloads/*
3. *Install Python Development Plugin in Eclipse: http://marketplace.eclipse.org/content/pydev-python-ide-eclipse*
4. **Download and install PyCharm**: https://www.jetbrains.com/pycharm/download/


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

PyDev - Python IDE for Eclipse 7.1.0

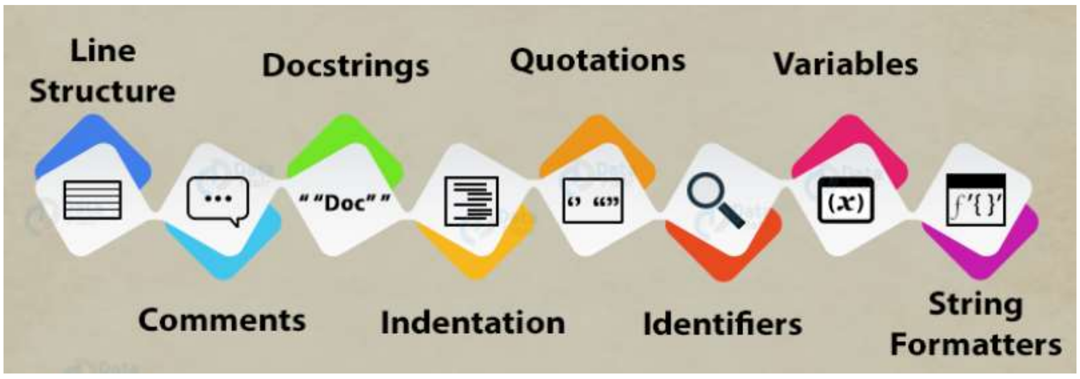
PyDev is a plugin that enables Eclipse to be used as a Python IDE (supporting also Jython and IronPython). It uses advanced type inference techniques which allow... **more info**

by Brainwy Software, EPL

IDE Python Aptana Pydev Django ...

★ 1359 Installs: **1.11M** (20,162 last month) Installed

Python Syntax



A Python program comprises logical lines. A **NEWLINE** token follows each of those. The interpreter ignores blank lines.

Indentations

Python uses indentation to indicate a block of code. Python will give you an error if you skip the indentation.

```
if 100 > 20:
    print("100 is Bigger than 20")
```

Comments:

Comments start with a #.

Eg. # You Cannot skip Indentations

Docstring:

- Python also has extended documentation capability, called docstrings.
- Docstrings can be one line, or multiline.
- Python uses triple quotes at the beginning and end of the docstring:

```
""" Now this String is Called DocString
Which can be Multiline also """
```

Variables

Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

variableName = value;

```
age = 5
name = "John"
print(age)
print(name)
```

Tip: Your variable name indicates your programming experience. Always use elaborated, camelCase variables.

A variable can have a short name (like x and y) or a more descriptive name (age, carName, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

There are three numeric types in Python:

- int
- float
- complex

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
- Float, or "floating point number" is a number, positive or negative, containing one or more decimals
- Complex numbers are written with a "j" as the imaginary part.

Casting

There may be times when you want to specify a type on to a variable. This can be done with casting.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
x = str("s1") # x will be 's1'

y = int(2.8) # y will be 2

z = float("3")   # z will be 3.0
```

Python Strings

A Python string is a sequence of characters. There is a built-in class 'str' for handling Python string. String literals in python are surrounded by either single quotation marks, or double quotation marks. Strings in Python are arrays of bytes representing unicode characters.

'hello' is the same as "hello".

```
a = "Hello, World!"

print(a[1])

print(a[2:5])

print(a.strip()) # removes any whitespace from the beginning or the end!"

print(len(a)) #returns the length of a string

print(a.lower()) # returns the string in lower case

print(a.upper()) #returns the string in upper case

print(a.replace("H", "J")) # replaces a string with another string

print(a.split(",")) # returns ['Hello', ' World!']

a.isdigit() # Returns True if all characters in a string are digits.

a.isalpha() # Returns True if all characters in a string are characters from an alphabet.

a.isspace() # Returns True if all characters in a string are spaces.
```

Python allows for command line input.

That means we are able to ask the user for input.

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Since we delimit strings using quotes, there are some things you need to take care of when using them inside a string.

```
a="Dogs are "love""
```

Concatenation is the operation of joining stuff together. Python Strings can join using the concatenation operator +.

```
a='Do you see this, '
b='$$?'
print(a+b)
```

The letter ‘f’ precedes the string, and the variables are mentioned in curly braces in their places.

```
name = "Ayushi"
day = "today"
print(f"Its not {name} 's first birthday {day}")
```

Python Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y

/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif

The `elif` keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In this example `a` is equal to `b`, so the first condition is not true, but the `elif` condition is true, so we print to screen that "a and b are equal".

Else

The `else` keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example `a` is greater than `b`, so the first condition is not true, also the `elif` condition is not true, so we go to the `else` condition and print to screen that "a is greater than b".

And

The `and` keyword is a logical operator, and is used to combine conditional statements:

Test if `a` is greater than `b`, AND if `c` is greater than `a`:

```
if a > b and c > a:
  print("Both conditions are True")
```

Or

The `or` keyword is a logical operator, and is used to combine conditional statements:

```
if a > b or a > c:
  print("At least one of the conditions is True")
```

Python Loops

Python has two primary loop commands:

- `while` loops

- `for` loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Print i as long as i is less than 6:

While condition:

        Block of code when condition is true

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Python For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The `for` loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

Exit the loop when `x` is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Using the range() function:

```
for x in range(6):
  print(x)
```

Note that `range(6)` is not the values of 0 to 6, but the values 0 to 5.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 6)`, which means values from 2 to 6 (but not including 6):

Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: `range(2, 30, 3)`:

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```