In python the space between the variables is automatically added which can be explained by the example given below:

Ex:

name\_person = “Steven”

print(“Hey my name is”, name\_person)

# Output of the program is Hey my name is Steven

Here the space before the name Steven is automatically added.

You cannot use reserved words as variables names/identifiers. The extension of the python files is “.py”.

Interactive v/s Script Python: A) In Interactive python you type directly to python one line at a time and it responds. B) In Script you enter a sequence of statements in to a file using a text editor and tell python to execute the statements in the file.

The way to print the variable inside a string is to put them inside the {} braces. It can be explained by the example given below:

Ex:

my\_name = ‘Zed A. Shaw’

my\_age = 35

my\_height = 74

my\_weight = 180

my\_eyes = ‘Blue’

my\_teeth = ‘White’

my\_hair = ‘Brown’

print(f”Let’s talk about {my\_name}.”)

print(f”He’s {my\_height} inches tall.”}

print(f”He’s {my\_weight} pounds heavy.”}

print(“Actually that’s not too heavy.”}

print(f”He’s got {my\_eyes} eyes and {my\_hair} hair.”)

print(f”His teeth are usually {my\_teeth} depending on the coffee.”)

total = my\_age + my\_height + my\_weight

print(f”If I add {my\_age}, {my\_height}, and {my\_weight} I get {total}.”)

Output:

Let’s talk about Zed A. Shaw.

He’s 74 inches tall.

He’s 180 pounds heavy.

Actually that’s not too heavy.

He’s got Blue eyes and Brown hair.

His teeth are usually White depending on the coffee.

If I add 35, 74, and 180 I get 289.

The variable name in python should always start with a character. So 1 is not a valid variable name while a1 is a valid variable name. We can use the round() function to round the floating point number.

In python it is typical to use single-quotes for any short strings like ‘a’ or ‘snow’.

Escape Sequences:

|  |  |
| --- | --- |
| Escape | What it does. |
| \\ | Backslash (\) |
| \’ | Single-quote (‘) |
| \” | Double-quote (“) |
| \a | ASCII bell (BEL) |
| \b | ASCII backspace (BS) |
| \f | ASCII formfeed (FF) |
| \n | ASCII linefeed (LF) |
| \N {name} | Character named name in the Unicode database (Unicode only) |
| \r | Carriage return (CR) |
| \t | Horizontal tab (TAB) |
| \uxxxx | Character with 16-bit hex value xxxx |
| \Uxxxxxxxx | Character with 32-bit hex value xxxxxxxx |
| \v | ASCII vertical tab (VT) |
| \ooo | Character with octal value 000 |
| \xhh | Character with hex value hh |

**Typecasting:**

1. Typecasting the input to Integer: There might be condition when you might require integer input form user/console, the following code takes two input(integer/float) from console and typecasts them to integer then prints the sum.

Ex:

Num1 = int(input())

Num2 = int(input())

#printing the sum in integer

print(Num1 + Num2)

1. Typecasting the input to Float: To covert the input to float the following code will work out.

Ex:

Num1 = float(input())

Num2 = float(input())

#printing the sum in float

print(Num1 + Num2)

1. Typecasting the input to String: All kind of input can be converted to string type whether they are float or integer. We make use of keyword str for typecasting.

Ex:

String = str(input())

#output

print(String)

Reading and Writing Files:

1. Read: Reads the contents of the file. You can assign the result to a variable.
2. Readline: Reads just one line of a text file.
3. Truncate: Empties the file. Watch out if you care about the file.
4. Write(‘stuff’): Writes “stuff” to the file.
5. Seek(0): Moves the read/write location to the beginning of the file.

When we want to interrupt or terminate the execution of the program while it is running in the python terminal then we should use combination of **CTRL + C** to execute it.

If we want the input() function to do nothing then we should press the **ENTER** button to execute the next line in our program.

There are many different ways to check if a file exists or not:

1. The exists() method in the os.path. Ex: os.path.exists(‘file\_ex.txt’)

If you want to ensure that a given path points to a file and not to a directory, you can use the os.path.isfile() function. Ex: os.path.isfile(‘file\_ex.txt’)

1. If the file exists the open call will complete successfully and return a valid file handle. If the file does not exist however, a FileNotFoundError exception will be raised.

Ex: try

F = open(‘myfile.txt’)

F.close()

except FileNotFoundError:

print(“File does not exist”)

1. Now the same “just attempt to open it” technique also works for ensuring a file is both readable and accessible. Instead of watching for FileNotFoundError exceptions you will want to look out for any kind of IOError:

Ex: try

F = open(“myfile.txt”)

F.close()

except IOError:

print(“File is not accessible”)

print(“File is accessible”)

1. If you frequently use this pattern you can factor it out into a helper function that will allow you to test whether a file exists and is accessible at the same time:

Ex: def is\_accessible(path, mode=’r’):

try:

f = open(path, mode)

f.close()

except IOError:

return False

return True

Alternatively, you can use the os.access() function in the standard library to check whether a file exists and is accessible at the same time. This would be more similar to using the os.path.exists() function for checking if a file exists.

Using open() and a try…except clause has some advantages when it comes to file handling in Python. It can help you avoid bugs caused by file existence race conditions.

It is recommended to use the file handling exception way to treat whether a file exists or not because, it will check right away if the file exists or not. While in using the exists() function if we check whether a file exists or not then we check first that the file exists() and it return True, but after executing the next line if the file is deleted then we will get an IOError which says that it is better to just directly open the file and using exception to handle whether the file exists or not.

In python when you have bytes and need a string decode bytes. When you have a string and need bytes encode strings.

Syntax: range([start], stop [, step])

Parameters:

Start: (optional) Starting point of the sequence. It defaults to 0.

Stop: (required) Endpoint of the sequence. This item will not be included in the sequence.

Step: (optional) Step size of the sequence. It defaults to 1.

The type() function is used to find the type of the input given to the type function.

Ex: type(eee)

<class’str’>

type(1)

<class’int’>

To convert an integer to floating point you can do ex: float(99) and it will be converted to 99.0

Integer division produces a floating point result.

String conversions: You can also use int() and float() to convert between strings and integers. You will get an error if the string does not contain numeric characters.

The keyword “is” is a strong keyword in python generally used to compare the equality of True, False and None. We can use it only for strong comparison or else we can use “==” sign.

Different import modules in Python are string, re, datetime, math, random, os, multiprocessing, subprocess, socket, email, json, doctest, unittest, pdb, argparse and sys

Different exceptions are raised for different reasons.

Common exceptions:

ImportError: an import fails

IndexError: a list is indexed with an out-of-range number

NameError: an unknown variable is used

SyntaxError: the code can’t be parsed properly

TypeError: a function is called on a value of an inappropriate type

ValueError: a function is called on a value of the correct type, but with an inappropriate value

In except block, the raise statement can be used without arguments to re-raise whatever exception occurred.

An assertion is a sanity-check that you can turn on or off when you have finished testing the program. An expression is tested, and if the results comes up false, an exception is raised. Assertions are carried out through the use of the assert statement. Programmers often place assertion at the start of a function to check for a valid input, and after a function call to check for the valid output.

AssertionError exception can be caught and handled like any other exception using the try-except statement, but if not handled, this type of exception will terminate the entire program.

To determine whether a key is in a dictionary, you can use in and not in, just as you can for a list.

A useful dictionary method is get. It does the same thing as indexing, but if the key is not found in the dictionary it returns another specified value instead(‘None’, by default).

Tuples are very similar to lists, except that they are immutable(they cannot be changed). Also, they are created using parentheses, rather than square brackets.

Tuples are faster than lists, but they cannot be changed.

List comprehensions are a useful way of quickly creating lists whose contents obey a simple rule.

Ex: cubes = [i\*\*3 for I in range(5)]

print(cubes)

The output is [0, 1, 8, 27, 64]

A list comprehension can also contain an if statement to enforce a condition or values in the list.

Ex: evens = [i\*\*2 for i in range(10) if i\*\*2 % 2 == 0]

print(evens)

The output is [0, 4, 16, 36, 64]

Functional programming seeks to use pure functions. Pure functions have no side effects, and return a value that depends only on their arguments.

In Python finite generator can be converted to the lists by passing them as arguments to the list function.

Using generators results in improved performance, which is the result of the lazy generation of values, which translates to lower memory usage. Furthermore, we do not need to wait until all the elements have been generated before we start to use them.

There are two ways you can iterate through a loop using the strings:

Ex:

1. fruit = ‘banana’

for letter in fruit :

print(letter)

1. index = 0

while index < len(fruit) :

letter = fruit[index]

print(letter)

index = index + 1

In python if the slice operation refers to an index of a string which is out of range then it will still not give an out of index error. For s=”Python” the output s[2:12] will give the output “thon”

If we leave off the first number of the last number of the slice, it is assumed to be the beginning or end of the string respectively.

Opening a File:

Before we can read the contents of the file, we must tell python which file we are going to work with and what we will be doing with the file. This is done with the open() function. Open() returns a “file handle” - a variable used to perform operations on the file. Similar to “File -> Open” in a word processor.

Handle = open(filename, mode)

When files are missing: When a file is not present the FileNotFoundError occurs in python.

The newline character: We use a special character called the “newline” to indicate when a line ends. We represent it as \n in strings. Newline is still one character- not two

Ex:

Stuff = ‘X\nY’

Here in the above code the len(Stuff) will be 3 and not 4 or 2.

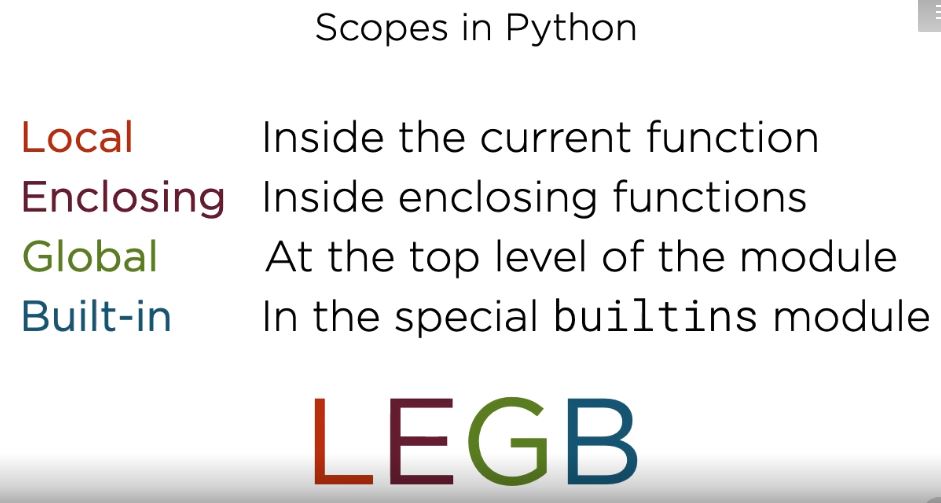
In python the print() statement always add the new line at the end of the statement.

A module is executed once on first import.

Arguments with default values must come after those without default values.

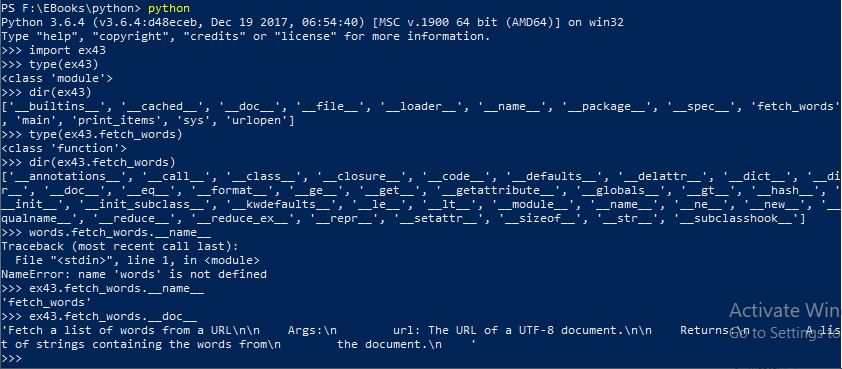
Always use immutable objects for default values.

Python will not generally perform implicit conversions between types.



To refer to the global variable for using in the function or any other purpose the global keyword is used before the variable name inside the function to access the global variable.

Everything is an object in python:



Python uses named references to objects.

Assignment attaches a name to an object.

Assigning one name to another makes them both point at the same object.

The garbage collector removes objects with no references.

id() returns a unique integer ID for an object.

Is determines if two names refer to the same object.

Function arguments are passed by object reference.

Rebinding function arguments loses the original object reference.

Return passes back an object reference to the caller.

Function arguments may have a default value.

Global references can be read from local scopes.

Use global to assign to global references from a local scope.

Everything in python is an object.

Import and def binds name to the object.

type() returns the type of an object.

dir() introspects the attributes of an object.

A single element with a trailing comma is parsed as a single element tuple.

To specify an empty tuple just write the empty parenthesis.

Tuple unpacking: Destructuring operation that unpacks data structures into named references.

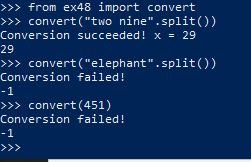
Strings:

In strings concatenation with + results in temporaries. str.join() inserts a separator between a collection of strings. Call join() on the separator string. To concatenate invoke join on empty text.

Dictionary:

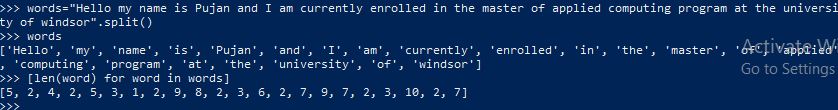
The keys in a dictionary must be immutable and the values in a dictionary may be mutable.

For Example ex48.py the output is given as:



Exception handling in python: Except-block can capture the exception. Avoid catching programming errors. Signal exceptional conditions with raise. raise without an argument re-raises the current exception. Generally don’t catch TypeError. Use str() to convert exceptions to strings. Prefer built-in exception types when possible. Use try…finally for cleanup actions. Implement platform-specific actions with ImportError and EAFP.

List comprehension syntax: [expr(item) for item in iterable]. The example can be found in the code snippet given below:



Dict comprehension syntax:

{

Key\_expr(item): value\_expr(item)

for item in iterable

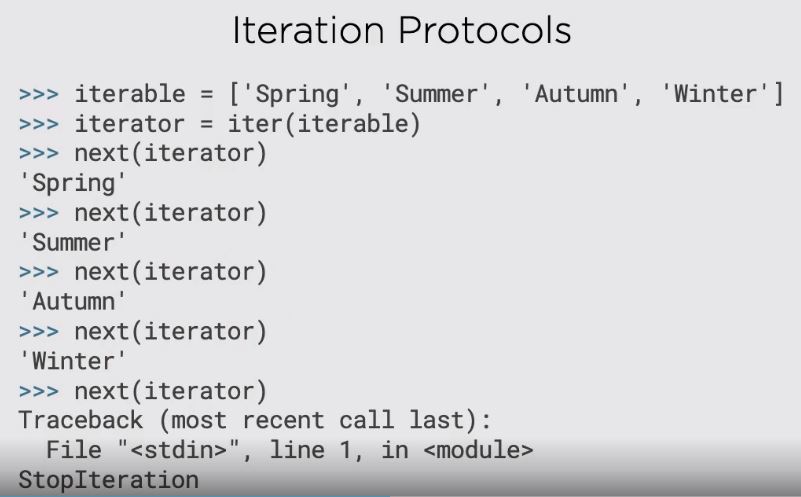
}

Iteration Protocols in python:

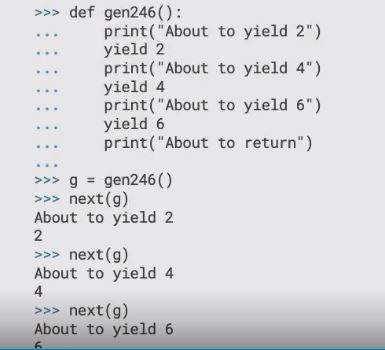
Iterable: Can be passed to iter() to produce an iterator

Iterator: Can be passed to next() to get the next value in the sequence

The repl activity is shown below:



The generator and yield in python can be explained by REPL below:



Generators only do enough work to produce requested data. This allows generators to model infinite (or just very large) sequences.

Generator Expressions are very similar to list comprehensions:

(expr(item) for item in iterable)

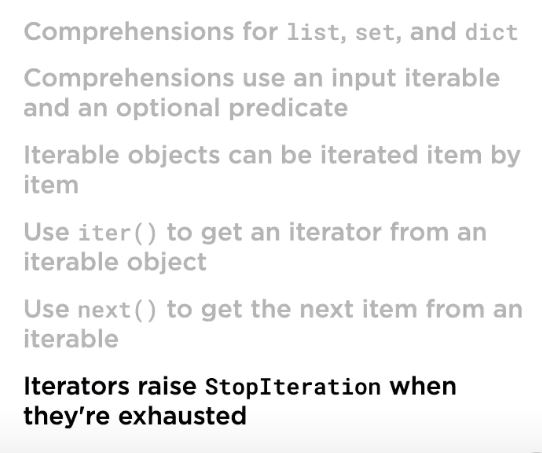
To recreate a generator from a generator expression, you must execute the expression again.

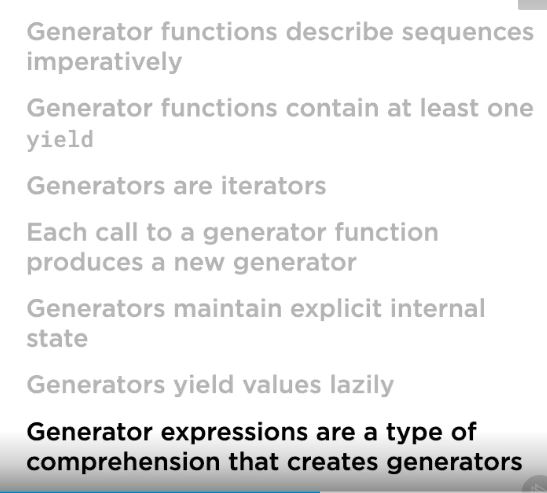
The itertools library provide many methods for iterating in a structure(list, set, dict, tuple) in Python.

The other important methods for iterators are:

1. any() – Determines If any elements in a series are true
2. all() – Determines if all elements in a series are true
3. zip() – Synchronize iteration across two or more iterables

Tips for Iterators:





Classes:

The first argument of instance methods in python always is self. \_\_init\_\_() is an initializer, not a constructor.

In packages are generally directories. Modules are generally files.

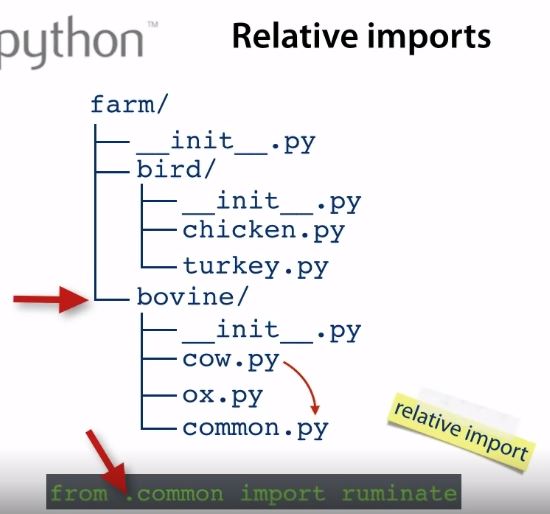
In python the path of each module is searched in the sys.path variable which contains the list of directories python searches for modules.

To add the third-party module in the python we can amend the sys.path to modify the path to include the directory in which the modules are present.

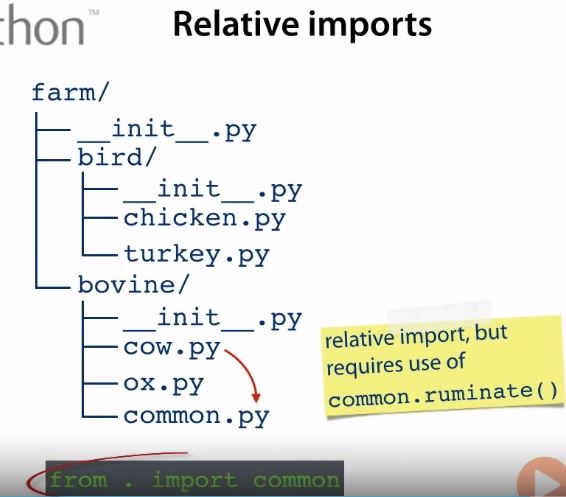
Packages:

1. Packages are modules that contains other modules
2. Packages are generally implemented as directories containing a special \_\_init\_\_.py file
3. The \_\_init\_\_.py file is executed when the package is imported
4. Packages can contain sub-packages which themselves are implemented with \_\_init\_\_.py files in directories

Relative Imports: The relative import can be understood by the example given below:



The other way we can import common module in cow.py is:



Generally, it is advisable to avoid the relative imports. The \_\_all\_\_ name is used in the \_\_init\_\_.py file to decide which modules to import when we are importing the packages using the \* like:

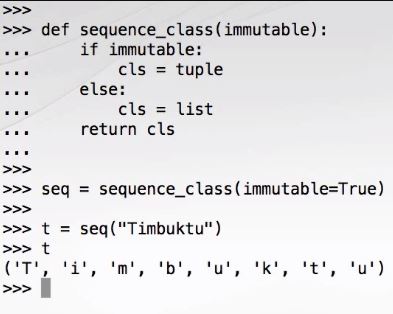
from module\_name import \*

Namespace packages have no \_\_init\_\_.py. This avoids complex initialization ordering problems.

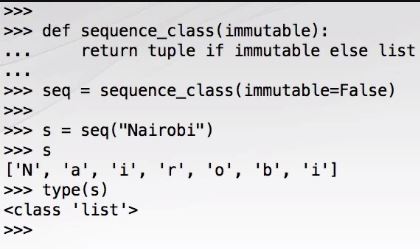
Importing namespace package:

1. Python scans all entries in sys.path
2. If a matching directory with \_\_init\_\_.py is found a normal package is loaded
3. If foo.py is found, then it is loaded
4. Otherwise, all matching directories in sys.path are considered as part of the namespace package

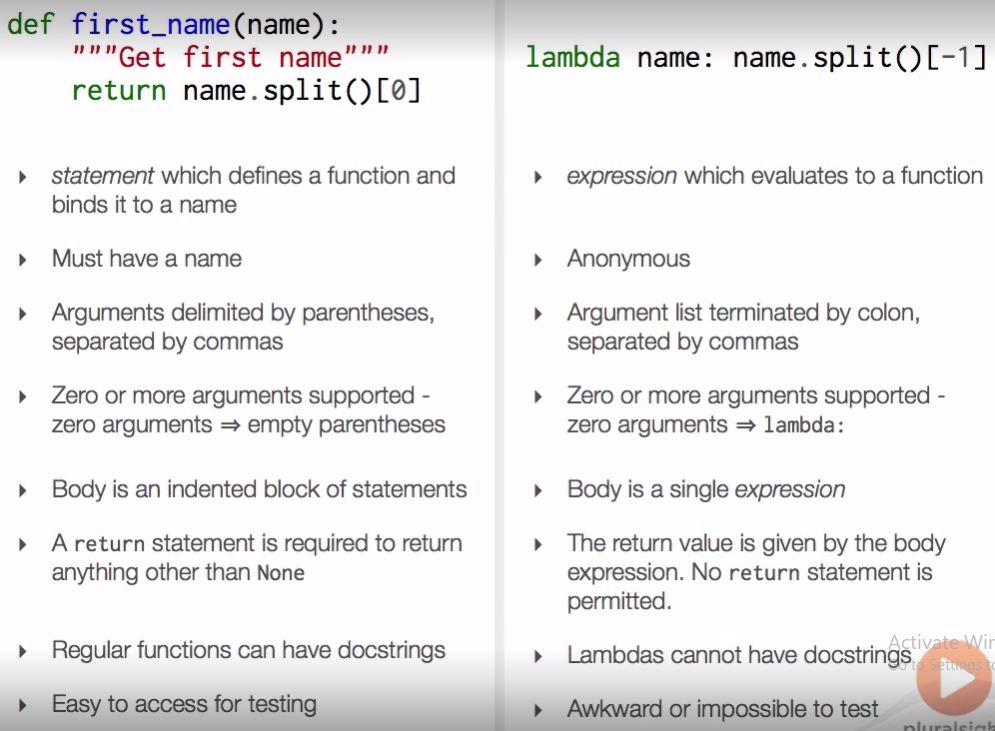
Classes are callable: In python classes are callable and the primary example of this feature is used when we invoke class to create a new object. The arguments that we use while instantiating an object in class can be used in the \_\_init\_\_ python function. This feature of python is explained in the code snippet give below:

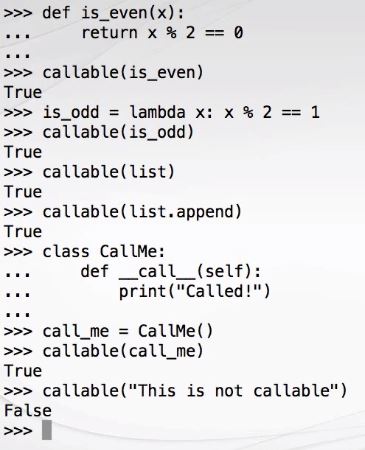


In python the conditional expressions are used frequently and can be given as below:



The difference between the lambda function and the normal function is given below:

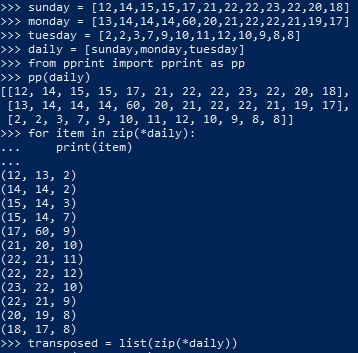


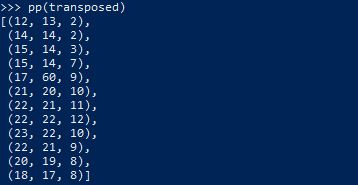


In functions the mandatory arguments must be passed before the optional arguments. In failure to do will result in the syntax error. All the features of the extended formal argument syntax applies to functions, lambdas and all other callables:

Syntax:- def extended(\*args, \*\*kwargs):

The asterisk can also be used on any iterable series as shown in the example below:





nonlocal: The nonlocal keyword introduces names form the enclosing namespace in to the local namespace. It searches form the innermost enclosing scope to the outermost enclosing scope.

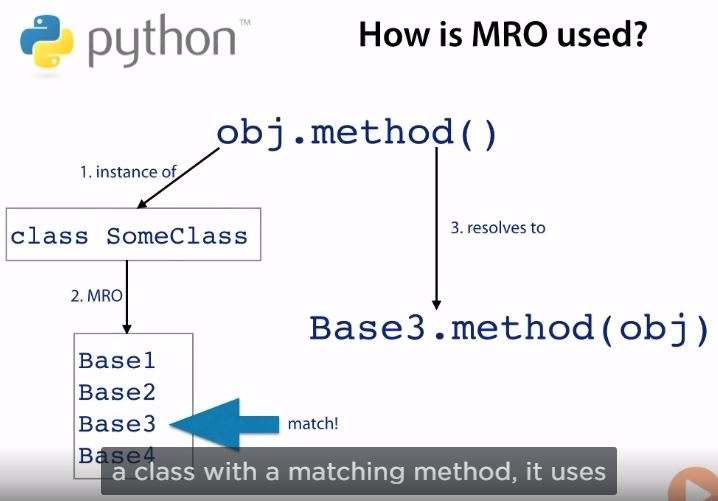
decorators: modify or enhance the functions without changing their definintion.

In python the validation of any field that is given in the setter method can also be used and triggered when we initialize an instance in python. So when we use the setter method for a particular field and write some code in it to modify or validate the variable behaviour it is automatically applied to the variable during the initialization that takes place when we create an object and assign the value in the \_\_init\_\_ constructor.

String Representation:

The built-in function repr() produces an unambiguous string representation of an object.

In python the Decimal and Fraction class are used for the floating numbers and the numbers in Fraction.



Subclasses inherit methods of all bases

Without conflict, names resolve in the obvious way

Method Resolution Order(MRO) determines name lookup in all cases

If a class has multiple base classes and defines no initializer

then only the initializer of the first base class is automatically called

The \_\_bases\_\_ method is associated with the name of the class to determine the name of the base classes of a particular class

For SortedIntList.\_\_mro\_\_ the resolution order class wise is given

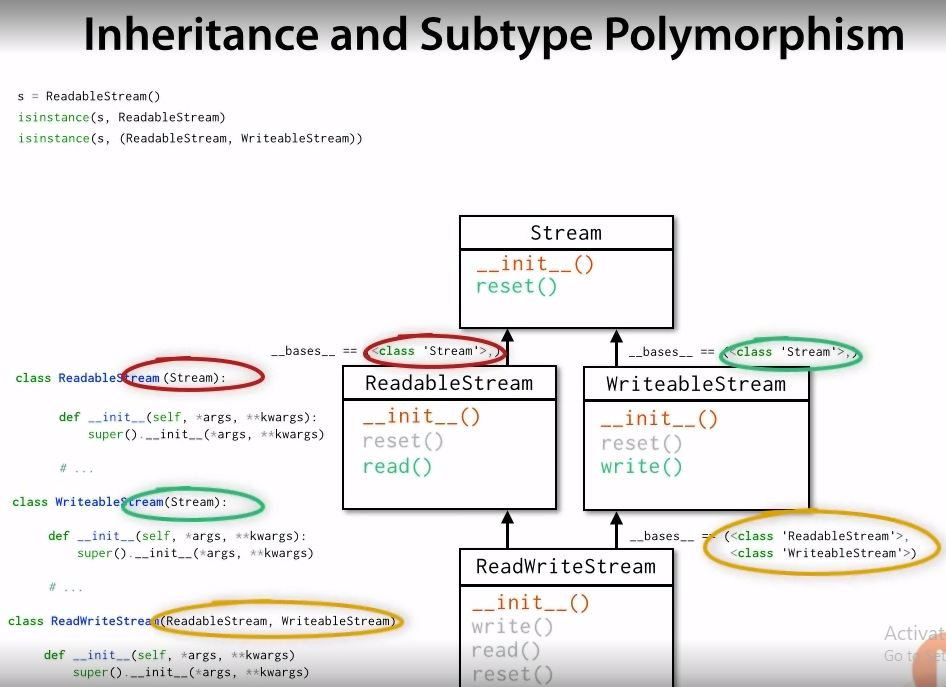
C3 – algorithm for calculating MRO in python

Subclasses come before base classes

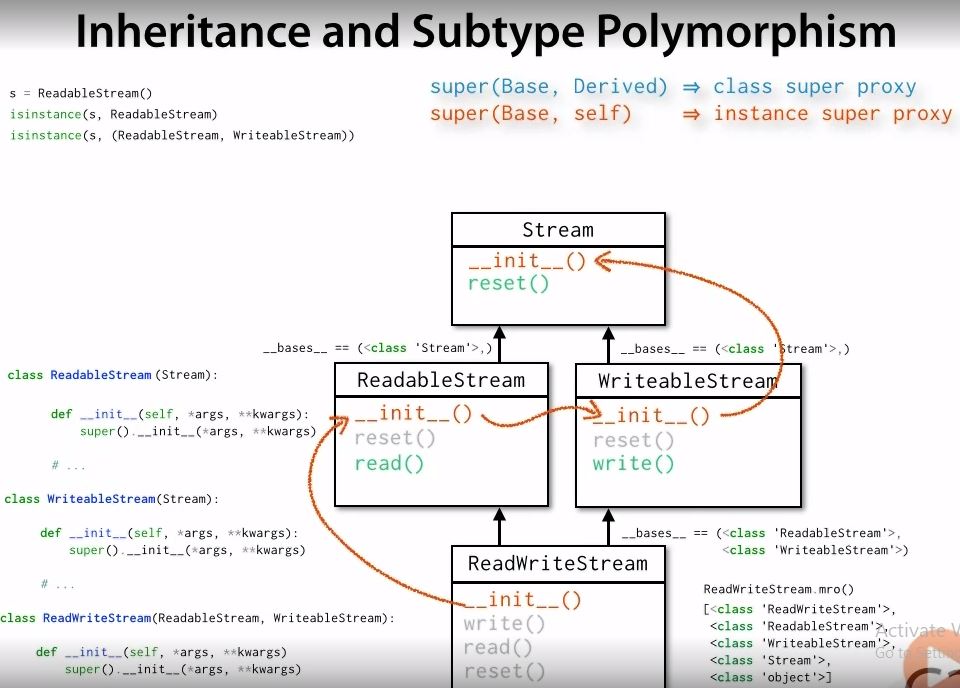
Base class order from class definition is preserved

First two qualities are preserved no matter where you start in the inheritance graph.

Given a method resolution order and a class C, super() gives you an object which resolves methods using only the part of the MRO which comes after C.



The output of mro() method for the above example and the type of proxy for the super method is:



Custom Collection:

The container protocol –

1. Membership testing using in and not in
2. Special method: \_\_contains\_\_(item)
3. Fallback to iterable protocol

The sized protocol –

1. Number of items using len(sized) function
2. Must not consume or modify collection
3. Special method: \_\_len\_\_()

The iterable protocol –

1. Obtain an iterator with iter(iterable) function
2. Special method: \_\_iter\_\_()

The sequence protocol –

1. Implies container, sized and iterable
2. Retrieve slices by slicing: item = seq[index]
3. Retrieve slices by slicing: item = seq[start:stop]
4. Special method \_\_getitem\_\_()
5. Find items by value: index = seq.index(item)
6. No special method
7. Concatenation with + operator. Special method \_\_add\_\_()
8. Repetition with \* operator. Special methods \_\_mul\_\_() and \_\_rmul\_\_()
9. Count items: num = seq.count(item). No special method
10. Produce a reversed sequence: r = reversed(seq). Special method \_\_reversed\_\_()
11. Fallback to \_\_getitem\_\_() and \_\_len\_\_()

Context manager: an object designed to be used in a with-statement

Ex: with context-manager:

Body

\_\_enter\_\_(): called before entering with-statement body

Return value bound to as variable

Can return value of any type

Commonly returns context-manager itself

\_\_exit\_\_(): called when with-statement body exits

Can check type for None to see if an exception was thrown

If \_\_exit\_\_() returns false, the exception is propagated.

\_\_exit\_\_() answers the question “should the with-statement swallow exceptions?”

Exceptions propagated form inner context managers will be seen by outer context managers

The pip command is important in python because it’s main aim is to install and uninstall any packages. For example: if we want to install requests python package then we can do that by the command: pip install requests. Similarly, if we want to uninstall the requests package, we can do that by the command: pip uninstall requests. But this is not a preferable way to install/uninstall any package as it makes the system wide changes so every program in your system runs on the version that pip installed for that package.

The preferred way to call pip is by explicitly mentioning the python version for which you want to install the module, cause in any system the pip install module for the python version that is configured in the system. To avoid any misunderstanding, it is preferred to install the module with the pip command along with the python version for which you want to install the version. Let’s say if you want to install the requests module for python 3.6.4 then you can do that by using the command: python 3.6.4 -m pip install requests

Problems with system-wide installs: multiple projects with conflicting dependencies, conflicts with system dependencies, multi-user systems, and testing code against different python and library versions.

Virtual Environments: Isolated context for installing packages. Always work inside a virtual environment. No global installs anymore. Create a virtual environment for every project. Isolate project dependencies.