Python

==================

Python is a high-level, interpreted, interactive and object-oriented scripting language.

Python is designed to be highly readable.

It uses English keywords frequently where as other languages use punctuation.

It has fewer syntactical constructions than other languages.

Python is a very simple language, and has a very straightforward syntax.

Python is object-oriented

Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

It's free (open source)

Downloading and installing Python is free and easy

Source code is easily accessible

It's portable

Python runs virtually every major platform used today

As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform.

Python is processed at runtime by the interpreter. You do not need to compile your program before executing it., like :PERL ,PHP.

Python is a great language for the beginner-level programmers as well as supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python-Execution

============

1. using python shell:

>>> print 'Hello world'

Hello world

2. using script file:

python script.py

Python versions:

There are two major Python versions

Python 2

and Python 3

Indentation

Python uses indentation for blocks, instead of curly braces.

Both tabs and spaces are supported, but the standard indentation requires standard Python code to use four spaces.

Variables and Types

Python is completely object oriented, and not "statically typed".

You do not need to declare variables before using them, or declare their type.

Every variable in Python is an object.

Python supports several different numeric types, LIKE:

Integers Examples: 0, 1, 1234, -56

Integers are implemented as C longs

Note: dividing an integer by another integer will return only the integer part of the quotient, e.g. typing 7/2 will yield 3

Long integers Example: 999999999999999999999L

Must end in either l or L ,Can be arbitrarily long

Floating point numbers Examples: 0., 1.0, 1e10, 3.14e-2, 6.99E4

Implemented as C doubles, Division works normally for floating point numbers: 7./2. = 3.5

NOTE: Operations involving both floats and integers will yield floats: Like: 6.4 – 2 = 4.4

Octal constants Examples: 0177, -01234

Must start with a leading 0

Hex constants Examples: 0x9ff, 0X7AE

Must start with a leading 0x or 0X

Complex numbers Examples: 3+4j, 3.0+4.0j, 2J

Must end in j or J, Typing in the imaginary part first will return the complex number in the order Re+ImJ

Lists:

Lists are very similar to arrays.

They can contain any type of variable, and they can contain as many variables as you wish.

Lists can also be iterated over in a very simple manner.

my\_list = ["apple", "banana", "cherry"]

print(my\_list)

Checking if an Item is in a List:

my\_list = ["apple", "banana", "cherry"]

print("banana" in my\_list) # Output: True

print("pineapple" in my\_list) # Output: False

print("banana" not in my\_list) # Output: False

print("pineapple" not in my\_list) # Output: True

Another way to check if an item is in a list is by using the count() method. This method returns the number of times the specified element appears in the list. If the count is more than 0, it means the item is in the list.

Here's an example:

my\_list = ["apple", "banana", "cherry"]

print(my\_list.count("banana")) # Output: 1

print(my\_list.count("pineapple")) # Output: 0

Python Operators:

================

Types and Operators: Operations on NumbersNumbers

Basic algebraic operations

Four arithmetic operations: a+b, a-b, a\*b, a/b

Exponentiation: a\*\*b

Comparison operators: Greater than, less than, etc.: a < b, a > b, a <= b, a >= b

Identity tests: a == b, a != b

Bitwise operators ============

Bitwise or: a | b

Bitwise exclusive or: a ^ b # Don't confuse this with exponentiation

Bitwise and: a & b

Shift a left or right by b bits: a << b, a >> b

Control Flow=============

if Statements

There can be zero or more elif parts, and the else part is optional.

The keyword ‘elif’ is short for ‘else if’, and is useful to avoid excessive indentation.

An if … elif … elif … sequence is a substitute for the switch or case statements found in other languages.

x = 3

y = 10

if x < y:

print("x is smaller than y.")

elif x == y:

print("x is equal to y.")

else:

print("x is greater than y.")

# Tomorrow is very hot

tomorrow = "very hot"

if tomorrow == "warm":

print("I'll go to the sea.")

elif tomorrow == "very hot":

print("I'll go to the forest.")

else:

print("I'll stay home.")

Loops

===========

l = ["ABC", "PQR", "XYZ"]

for i in l:

print(i)

s = "ABCDE"

for i in s:

print(i)

fruits = ["apple", "banana", "cherry"]

colors = ["red", "yellow", "green"]

for fruit, color in zip(fruits, colors):

print(fruit, "is", color)

# Prints all letters except 'e' and 's'

for letter in 'abcdeABCDE':

if letter == 'e' or letter == 'D':

continue

print('Current Letter :', letter)

"while" loops

==================

While loops repeat as long as a certain boolean condition is met.

while condition:

statment-1

break vs continue

====================

The break statement, like in C, breaks out of the innermost enclosing for or while loop.

Loop statements may have an else clause; it is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while), but not when the loop is terminated by a break statement.

break is used to exit a for loop or a while loop, whereas continue is used to skip the current block, and return to the "for" or "while" statement.

range() Function

=======================

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy.

range(5) -- go through 0 to 4

The given end point is never part of the generated sequence; range(10) generates 10 values, the legal indices for items of a sequence of length 10.

It is possible to let the range start at another number, or to specify a different increment :

range(5,10) --- go through 5 to 9

range(0,10,3) --- go through 0 3 6 9

range(-10,-100,-30) -- go through -10 -40 -70

Strings

============================

Strings are defined either with a single quote or a double quotes.

mystring = 'hello'

print(mystring)

mystring = "hello"

print(mystring)

The difference between the two is that using double quotes makes it easy to include apostrophes

mystring = "Don't worry about apostrophes"

print(mystring)

a. Strings are ordered blocks of text

b. Strings are enclosed in single or double quotation marks

c. Double quotation marks allow the user to extend strings over multiple lines without backslashes, which usually signal the continuation of an expression

Examples: 'abc', "ABC"

d. Concatenation and repetition

Strings are concatenated with the + sign:

>>> 'abc'+'def'

'abcdef'

Strings are repeated with the \* sign:

>>> 'abc'\*3

'abcabcabc'

Python string is a sequence of Unicode characters that is enclosed in quotation marks.

The below Python functions are used to change the case of the strings. Let’s look at some Python string methods with examples:

lower(): Converts all uppercase characters in a string into lowercase

upper(): Converts all lowercase characters in a string into uppercase

title(): Convert string to title case

swapcase(): Swap the cases of all characters in a string

capitalize(): Convert the first character of a string to uppercase

casefold() Implements caseless string matching

center() Pad the string with the specified character.

count() Returns the number of occurrences of a substring in the string.

encode() Encodes strings with the specified encoded scheme

endswith() Returns “True” if a string ends with the given suffix

expandtabs() Specifies the amount of space to be substituted with the “\t” symbol in the string

find() Returns the lowest index of the substring if it is found

format() Formats the string for printing it to console

format\_map() Formats specified values in a string using a dictionary

index() Returns the position of the first occurrence of a substring in a string

isalnum() Checks whether all the characters in a given string is alphanumeric or not

isalpha() Returns “True” if all characters in the string are alphabets

isdecimal() Returns true if all characters in a string are decimal

isdigit() Returns “True” if all characters in the string are digits

isidentifier() Check whether a string is a valid identifier or not

islower() Checks if all characters in the string are lowercase

isnumeric() Returns “True” if all characters in the string are numeric characters

isprintable() Returns “True” if all characters in the string are printable or the string is empty

isspace() Returns “True” if all characters in the string are whitespace characters

istitle() Returns “True” if the string is a title cased string

isupper() Checks if all characters in the string are uppercase

join() Returns a concatenated String

ljust() Left aligns the string according to the width specified

lower() Converts all uppercase characters in a string into lowercase

lstrip() Returns the string with leading characters removed

maketrans() Returns a translation table

partition() Splits the string at the first occurrence of the separator

replace() Replaces all occurrences of a substring with another substring

rfind() Returns the highest index of the substring

rindex() Returns the highest index of the substring inside the string

rjust() Right aligns the string according to the width specified

rpartition() Split the given string into three parts

rsplit() Split the string from the right by the specified separator

rstrip() Removes trailing characters

splitlines() Split the lines at line boundaries

startswith() Returns “True” if a string starts with the given prefix

strip() Returns the string with both leading and trailing characters

swapcase() Converts all uppercase characters to lowercase and vice versa

title() Convert string to title case

translate() Modify string according to given translation mappings

zfill() Returns a copy of the string with ‘0’ characters padded to the left side of the string

Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time.

Also functions are a key way to define interfaces so programmers can share their code.

How do you write functions in Python?

------------------------------------

Functions in python are defined using the block keyword "def", followed with the function's name as the block's name.

Functions may also receive arguments (variables passed from the caller to the function).

Functions may return a value to the caller, using the keyword- 'return' .

Simply write the function's name followed by (), placing any required arguments within the brackets – To Call a function.

Usually, function definitions have the following basic structure:

def func(args):

return values

Regardless of the arguments, (including the case of no arguments) a function call must end with parentheses.

Functions may be simple one-to-one mappings

>>> def f1(x):

... return x\*(x-1)

...

To execute function

>>> f1(3)

6

Function with multiple input and/or output variables

>>> def f2(x,y):

... return x+y,x-y

...

To execute:

>>> f2(3,2)

(5,1)

Function with multiple input and/or output variables

>>> def f2(x,y):

... return x+y,x-y

...

To execute:

>>> f2(3,2)

(5,1)

Functions don't need to contain arguments at all:

>>> def f3():

... print 'Hello world'

...

>>> f3()

Hello world

The user can set arguments to default values in function definitions:

>>> def f4(x,a=1):

... return a\*x\*\*2

...

>>>

If this function is called with only one argument, the default value of 1 is assumed for the second argument

>>> f4(2)

4

However, the user is free to change the second argument from its default value

>>> f4(2,a=2) # f4(2,2) would also work

8

Functions need not take just numbers as arguments, nor output just numbers or tuples. Rather, they can take multiple types as inputs and/or outputs.

Examples:

>>> arr = arange(4)

>>> f4(arr,a=2) # using the same f4 as on the previous slide

[0,2,8,18,]

>>> def f5(func, list, x):

... L = []

... for i in range(len(list)):

... L.append(func(x+list[i]))

... arr = array(L)

... return L,arr

...

>>> L1 = [0.0,0.1,0.2,0.3]

>>> L,arr = f5(exp,L1,0.5)

>>> arr

[ 1.64872127, 1.8221188 , 2.01375271, 2.22554093,]

Note: the function above requires Numeric, NumPy, or a similar package

Anything calculated inside a function but not specified as an output quantity (either with return or global) will be deleted once the function stops running

>>> def f5(x,y):

... a = x+y

... b = x-y

... return a\*\*2,b\*\*2

...

>>> f5(3,2)

(25,1)

If we try to call a or b, we get an error message:

>>> a

Traceback (most recent call last):

File "<stdin>", line 1, in ?

NameError: name 'a' is not defined

This brings us to scoping issues, which will be addressed in the next section.

Python string is a sequence of Unicode characters that is enclosed in quotation marks.

The below Python functions are used to change the case of the strings. Let’s look at some Python string methods with examples:

lower(): Converts all uppercase characters in a string into lowercase

upper(): Converts all lowercase characters in a string into uppercase

title(): Convert string to title case

swapcase(): Swap the cases of all characters in a string

capitalize(): Convert the first character of a string to uppercase

casefold() Implements caseless string matching

center() Pad the string with the specified character.

count() Returns the number of occurrences of a substring in the string.

encode() Encodes strings with the specified encoded scheme

endswith() Returns “True” if a string ends with the given suffix

expandtabs() Specifies the amount of space to be substituted with the “\t” symbol in the string

find() Returns the lowest index of the substring if it is found

format() Formats the string for printing it to console

format\_map() Formats specified values in a string using a dictionary

index() Returns the position of the first occurrence of a substring in a string

isalnum() Checks whether all the characters in a given string is alphanumeric or not

isalpha() Returns “True” if all characters in the string are alphabets

isdecimal() Returns true if all characters in a string are decimal

isdigit() Returns “True” if all characters in the string are digits

isidentifier() Check whether a string is a valid identifier or not

islower() Checks if all characters in the string are lowercase

isnumeric() Returns “True” if all characters in the string are numeric characters

isprintable() Returns “True” if all characters in the string are printable or the string is empty

isspace() Returns “True” if all characters in the string are whitespace characters

istitle() Returns “True” if the string is a title cased string

isupper() Checks if all characters in the string are uppercase

join() Returns a concatenated String

ljust() Left aligns the string according to the width specified

lower() Converts all uppercase characters in a string into lowercase

lstrip() Returns the string with leading characters removed

maketrans() Returns a translation table

partition() Splits the string at the first occurrence of the separator

replace() Replaces all occurrences of a substring with another substring

rfind() Returns the highest index of the substring

rindex() Returns the highest index of the substring inside the string

rjust() Right aligns the string according to the width specified

rpartition() Split the given string into three parts

rsplit() Split the string from the right by the specified separator

rstrip() Removes trailing characters

splitlines() Split the lines at line boundaries

startswith() Returns “True” if a string starts with the given prefix

strip() Returns the string with both leading and trailing characters

swapcase() Converts all uppercase characters to lowercase and vice versa

title() Convert string to title case

translate() Modify string according to given translation mappings

zfill() Returns a copy of the string with ‘0’ characters padded to the left side of the string

Python Operator Overloading

===================

In Python, we can change the way operators work for user-defined types.

For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.

Advantages of Operator Overloading

Here are some advantages of operator overloading,

1. Improves code readability by allowing the use of familiar operators.

2. Ensures that objects of a class behave consistently with built-in types and other user-defined types.

3. Makes it simpler to write code, especially for complex data types.

4. Allows for code reuse by implementing one operator method and using it for other operators.

=====To overload the + operator, we will need to implement \_\_add\_\_() function in the class.

With great power comes great responsibility. We can do whatever we like inside this function. But it is more sensible to return the Point object of the coordinate sum.

class Point:

def \_\_init\_\_(self, x=0, y=0):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "({0},{1})".format(self.x, self.y)

def \_\_add\_\_(self, other):

x = self.x + other.x

y = self.y + other.y

return Point(x, y)

p1 = Point(1, 2)

p2 = Point(2, 3)

print(p1+p2)

# Output: (3,5)

==another example for + overload

# Python Program to perform addition

# of two complex numbers using binary

# + operator overloading.

class complex:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

# adding two objects

def \_\_add\_\_(self, other):

return self.a + other.a, self.b + other.b

Ob1 = complex(1, 2)

Ob2 = complex(2, 3)

Ob3 = Ob1 + Ob2

print(Ob3)

Similarly, we can overload other operators as well. The special function that we need to implement is tabulated below.

Operator Expression Internally

Addition p1 + p2 p1.\_\_add\_\_(p2)

Subtraction p1 - p2 p1.\_\_sub\_\_(p2)

Multiplication p1 \* p2 p1.\_\_mul\_\_(p2)

Power p1 \*\* p2 p1.\_\_pow\_\_(p2)

Division p1 / p2 p1.\_\_truediv\_\_(p2)

Floor Division p1 // p2 p1.\_\_floordiv\_\_(p2)

Remainder (modulo)

p1 % p2 p1.\_\_mod\_\_(p2)

Bitwise Left Shift

p1 << p2 p1.\_\_lshift\_\_(p2)

Bitwise Right Shift

p1 >> p2 p1.\_\_rshift\_\_(p2)

Bitwise AND p1 & p2 p1.\_\_and\_\_(p2)

Bitwise OR p1 | p2 p1.\_\_or\_\_(p2)

Bitwise XOR p1 ^ p2 p1.\_\_xor\_\_(p2)

Bitwise NOT ~p1 p1.\_\_invert\_\_()

# Overloading ~ operator, but with two operands

================================================

# Python program which attempts to

# overload ~ operator as binary operator

class A:

def \_\_init\_\_(self, a):

self.a = a

# Overloading ~ operator, but with two operands

def \_\_invert\_\_(self):

return "This is the ~ operator, overloaded as binary operator."

ob1 = A(2)

print(~ob1)

Overloading Comparison Operators

==============================

Python does not limit operator overloading to arithmetic operators only. We can overload comparison operators as well.

Here's an example of how we can overload the < operator to compare two objects the Person class based on their age:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# overload < operator

def \_\_lt\_\_(self, other):

return self.age < other.age

p1 = Person("Alice", 20)

p2 = Person("Bob", 30)

print(p1 < p2) # prints True

print(p2 < p1) # prints False

Similarly, the special functions that we need to implement, to overload other comparison operators are tabulated below.

Operator Expression Internally

Less than p1 < p2 p1.\_\_lt\_\_(p2)

Less than or equal to p1 <= p2 p1.\_\_le\_\_(p2)

Equal to p1 == p2 p1.\_\_eq\_\_(p2)

Not equal to p1 != p2 p1.\_\_ne\_\_(p2)

Greater than p1 > p2 p1.\_\_gt\_\_(p2)

Greater than or equal to p1 >= p2 p1.\_\_ge\_\_(p2)

Python - Public, Protected, Private Members

===============

Python uses ‘\_’ symbol to determine the access control for a specific data member or a member function of a class. Access specifiers in Python have an important role to play in securing data from unauthorized access and in preventing it from being exploited.

A Class in Python has three types of access modifiers:

Public Access Modifier

Protected Access Modifier

Private Access Modifier

Public Members

Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method. This arrangement of private instance variables and public methods ensures the principle of data encapsulation.

All members in a Python class are public by default. Any member can be accessed from outside the class environment.

class Student:

schoolName = 'XYZ School' # class attribute

def \_\_init\_\_(self, name, age):

self.name=name # instance attribute

self.age=age # instance attribute

You can access the Student class's attributes and also modify their values, as shown below.

std = Student("Steve", 25)

print(std.schoolName) #'XYZ School'

print(std.name) #'Steve'

std.age = 20

print(std.age)

Protected Members

Protected members of a class are accessible from within the class and are also available to its sub-classes. No other environment is permitted access to it. This enables specific resources of the parent class to be inherited by the child class.

Python's convention to make an instance variable protected is to add a prefix \_ (single underscore) to it. This effectively prevents it from being accessed unless it is from within a sub-class.

class Student:

\_schoolName = 'XYZ School' # protected class attribute

def \_\_init\_\_(self, name, age):

self.\_name=name # protected instance attribute

self.\_age=age # protected instance attribute

In fact, this doesn't prevent instance variables from accessing or modifying the instance. You can still perform the following operations:

std = Student("Swati", 25)

print(std.\_name) #'Swati'

std.\_name = 'Dipa'

print(std.\_name) #'Dipa'

However, you can define a property using property decorator and make it protected, as shown below.

class Student:

def \_\_init\_\_(self,name):

self.\_name = name

@property

def name(self):

return self.\_name

@name.setter

def name(self,newname):

self.\_name = newname

Above, @property decorator is used to make the name() method as property and @name.setter decorator to another overloads of the name() method as property setter method. Now, \_name is protected.

std = Student("Swati")

print(std.name) #'Swati'

std.name = 'Dipa'

print(std.name) #'Dipa'

print(std.\_name) #'Dipa'

Above, we used std.name property to modify \_name attribute. However, it is still accessible in Python. Hence, the responsible programmer would refrain from accessing and modifying instance variables prefixed with \_ from outside its class.

Private Members

Python doesn't have any mechanism that effectively restricts access to any instance variable or method. Python prescribes a convention of prefixing the name of the variable/method with a single or double underscore to emulate the behavior of protected and private access specifiers.

The double underscore \_\_ prefixed to a variable makes it private. It gives a strong suggestion not to touch it from outside the class. Any attempt to do so will result in an AttributeError:

class Student:

\_\_schoolName = 'XYZ School' # private class attribute

def \_\_init\_\_(self, name, age):

self.\_\_name=name # private instance attribute

self.\_\_salary=age # private instance attribute

def \_\_display(self): # private method

print('This is private method.')

std = Student("Bill", 25)

print(std.\_\_schoolName) #AttributeError

print(std.\_\_name) #AttributeError

print(std.\_\_display()) #AttributeError

Python performs name mangling of private variables. Every member with a double underscore will be changed to \_object.\_class\_\_variable. So, it can still be accessed from outside the class, but the practice should be refrained.

std = Student("Bill", 25)

print(std.\_Student\_\_name) #'Bill'

std.\_Student\_\_name = 'Steve'

print(std.\_Student\_\_name) #'Steve'

std.\_Student\_\_display() #'This is private method.'

++++++++++

Thus, Python provides conceptual implementation of public, protected, and private access modifiers, but not like other languages like C#, Java, C++.

Introduction to Python Object-oriented Programming

=================================================

Everything in Python is an object.

An object has a state and behaviors.

To create an object, you define a class first.

And then, from the class, you can create one or more objects. The objects are instances of a class.

Define a class

To define a class, you use the class keyword followed by the class name. For example, the following defines a Person class:

class Person:

To create an object from the Person class, you use the class name followed by parentheses (), like calling a function:

person = Person()

Define instance attributes

=============================

Python is dynamic. It means that you can add an attribute to an instance of a class dynamically at runtime.

For example, the following adds the name attribute to the person object:

person.name = 'John'

However, if you create another Person object, the new object won’t have the name attribute.

To define and initialize an attribute for all instances of a class, you use the \_\_init\_\_ method.

The following defines the Person class with two instance attributes name and age:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

When you create a Person object, Python automatically calls the \_\_init\_\_ method to initialize the instance attributes.

In the \_\_init\_\_ method, the self is the instance of the Person class.

The following creates a Person object named person:

person = Person('John', 25)

Define instance methods

=================

The following adds an instance method called greet() to the Person class:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def greet(self):

return f"Hi, it's {self.name}."

To call an instance method, you also use the dot notation.

For example:

person = Person('John', 25)

print(person.greet())

Define class attributes

=====================

Unlike instance attributes, class attributes are shared by all instances of the class.

They are helpful if you want to define class constants or variables that keep track of the number of instances of a class.

For example, the following defines the counter class attribute in the Person class:

class Person:

counter = 0

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def greet(self):

return f"Hi, it's {self.name}."

You can access the counter attribute from the Person class:

Person.counter

To make the counter variable more useful, you can increase its value by one once an object is created. To do it, you increase the counter class attribute in the \_\_init\_\_ method:

class Person:

counter = 0

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Person.counter += 1

def greet(self):

return f"Hi, it's {self.name}."

The following creates two instances of the Person class and shows the value of the counter:

p1 = Person('John', 25)

p2 = Person('Jane', 22)

print(Person.counter)

Output:

2

Define class method

==============

Like a class attribute, a class method is shared by all instances of the class.

The first argument of a class method is the class itself.

By convention, its name is cls.

Python automatically passes this argument to the class method. Also, you use the @classmethod decorator to decorate a class method.

class Person:

counter = 0

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Person.counter += 1

def greet(self):

return f"Hi, it's {self.name}."

@classmethod

def create\_anonymous(cls):

return Person('Anonymous', 22)

The following shows how to call the create\_anonymous() class method:

anonymous = Person.create\_anonymous()

print(anonymous.name) # Anonymous

Define static method

===================

A static method is not bound to a class or any instances of the class.

In Python, you use static methods to group logically related functions in a class.

To define a static method, you use the @staticmethod decorator.

For example, the following defines a class TemperatureConverter that has two static methods that convert from celsius to Fahrenheit and vice versa:

class TemperatureConverter:

@staticmethod

def celsius\_to\_fahrenheit(c):

return 9 \* c / 5 + 32

@staticmethod

def fahrenheit\_to\_celsius(f):

return 5 \* (f - 32) / 9

To call a static method, you use the ClassName.static\_method\_name() syntax. For example:

f = TemperatureConverter.celsius\_to\_fahrenheit(30)

print(f) # 86

Code language: Python (python)

Notice that Python doesn’t implicitly pass an instance (self) as well as class (cls) as the first argument of a static method.

Single inheritance

=========================

A class can reuse another class by inheriting it.

When a child class inherits from a parent class, the child class can access the attributes and methods of the parent class.

For example, you can define an Employee class that inherits from the Person class:

class Employee(Person):

def \_\_init\_\_(self, name, age, job\_title):

super().\_\_init\_\_(name, age)

self.job\_title = job\_title

Inside the \_\_init\_\_ method of the Employee class calls the \_\_init\_\_method of the Person class to initialize the name and age attributes.

The super() allows a child class to access a method of the parent class.

The Employee class extends the Person class by adding one more attribute called job\_title.

The Person is the parent class while the Employee is a child class.

To override the greet() method in the Person class, you can define the greet() method in the Employee class as follows:

class Employee(Person):

def \_\_init\_\_(self, name, age, job\_title):

super().\_\_init\_\_(name, age)

self.job\_title = job\_title

def greet(self):

return super().greet() + f" I'm a {self.job\_title}."

The greet() method in the Employee is also called the greet() method of the Person class.

In other words, it delegates to a method of the parent class.

The following creates a new instance of the Employee class and call the greet() method:

employee = Employee('John', 25, 'Python Developer')

print(employee.greet())

Output:

Hi, it's John. I'm a Python Developer.

Introduction to Python overridding method

=====================

The overriding method allows a child class to provide a specific implementation of a method that is already provided by one of its parent classes.

First, define the Employee class:

class Employee:

def \_\_init\_\_(self, name, base\_pay):

self.name = name

self.base\_pay = base\_pay

def get\_pay(self):

return self.base\_pay

The Employee class has two instance variables name and base\_pay. It also has the get\_pay() method that returns the base\_pay.

Second, define the SalesEmployee that inherits from the Employee class:

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

The SalesEmployee class has three instance attributes: name, base\_pay, and sales\_incentive.

Third, create a new instance of the SalesEmployee class and display the pay:

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

Output:

5000

The get\_pay() method returns only the base\_pay, not the sum of the base\_pay and sales\_incentive.

When you call the get\_pay() from the instance of the SalesEmployee class, Python executes the get\_pay() method of the Employee class, which returns the base\_pay.

To include the sales incentive in the pay, you need to redefine the get\_pay() method in the SalesEmployee class as follows:

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

def get\_pay(self):

return self.base\_pay + self.sales\_incentive

In this case, we say that the get\_pay() method in the SalesEmployee class overrides the get\_pay() method in the Employee class.

When you call the get\_pay() method of the SalesEmployee‘s object, Python will call the get\_pay() method in the SalesEmployee class:

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

Output:

6500

If you create an instance of the Employee class, Python will call the get\_pay() method of the Employee class, not the get\_pay() method of the SalesEmployee class. For example:

jane = Employee('Jane', 5000)

print(jane.get\_pay())

FULL CODE:

class Employee:

def \_\_init\_\_(self, name, base\_pay):

self.name = name

self.base\_pay = base\_pay

def get\_pay(self):

return self.base\_pay

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

def get\_pay(self):

return self.base\_pay + self.sales\_incentive

if \_\_name\_\_ == '\_\_main\_\_':

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

jane = Employee('Jane', 5000)

print(jane.get\_pay())

Introduction to Python Abstract Classes

In object-oriented programming, an abstract class is a class that cannot be instantiated. However, you can create classes that inherit from an abstract class.

Typically, you use an abstract class to create a blueprint for other classes.

Similarly, an abstract method is an method without an implementation. An abstract class may or may not include abstract methods.

Python doesn’t directly support abstract classes. But it does offer a module that allows you to define abstract classes.

To define an abstract class, you use the abc (abstract base class) module.

The abc module provides you with the infrastructure for defining abstract base classes.

For example:

from abc import ABC

class AbstractClassName(ABC):

To define an abstract method, you use the @abstractmethod decorator:

from abc import ABC, abstractmethod

class AbstractClassName(ABC):

@abstractmethod

def abstract\_method\_name(self):

Python abstract class example

from abc import ABC, abstractmethod

class Employee(ABC):

def \_\_init\_\_(self, first\_name, last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

@property

def full\_name(self):

return f"{self.first\_name} {self.last\_name}"

@abstractmethod

def get\_salary(self):

pass

class FulltimeEmployee(Employee):

def \_\_init\_\_(self, first\_name, last\_name, salary):

super().\_\_init\_\_(first\_name, last\_name)

self.salary = salary

def get\_salary(self):

return self.salary

class HourlyEmployee(Employee):

def \_\_init\_\_(self, first\_name, last\_name, worked\_hours, rate):

super().\_\_init\_\_(first\_name, last\_name)

self.worked\_hours = worked\_hours

self.rate = rate

def get\_salary(self):

return self.worked\_hours \* self.rate

class Payroll:

def \_\_init\_\_(self):

self.employee\_list = []

def add(self, employee):

self.employee\_list.append(employee)

def print(self):

for e in self.employee\_list:

print(f"{e.full\_name} \t ${e.get\_salary()}")

The main program

The following app.py uses the FulltimeEmployee, HourlyEmployee, and Payroll classes to print out the payroll of five employees.

from fulltimeemployee import FulltimeEmployee

from hourlyemployee import HourlyEmployee

from payroll import Payroll

payroll = Payroll()

payroll.add(FulltimeEmployee('John', 'Doe', 6000))

payroll.add(FulltimeEmployee('Jane', 'Doe', 6500))

payroll.add(HourlyEmployee('Jenifer', 'Smith', 200, 50))

payroll.add(HourlyEmployee('David', 'Wilson', 150, 100))

payroll.add(HourlyEmployee('Kevin', 'Miller', 100, 150))

payroll.print()

Output:

John Doe $6000

Jane Doe $6500

Jenifer Smith $10000

David Wilson $15000

Kevin Miller $15000

Dictionaries in Python

====================

Python provides another composite data type called a dictionary, which is similar to a list in that it is a collection of objects.

Dictionaries and lists share the following characteristics:

Both are mutable.

Both are dynamic. They can grow and shrink as needed.

Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries differ from lists primarily in how elements are accessed:

List elements are accessed by their position in the list, via indexing.

Dictionary elements are accessed via keys.

Defining a Dictionary===========

d = {

<key>: <value>,

<key>: <value>,

.

.

.

<key>: <value>

}

Create==========

>>> MLB\_team = dict(

... Colorado='Rockies',

... Boston='Red Sox',

... Minnesota='Twins',

... Milwaukee='Brewers',

... Seattle='Mariners'

... )

>>> type(MLB\_team)

<class 'dict'>

>>> MLB\_team

{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins',

'Milwaukee': 'Brewers', 'Seattle': 'Mariners'}

Adding an entry to an existing dictionary is simply a matter of assigning a new key and value:

>>> MLB\_team['Kansas City'] = 'Royals'

If you want to update an entry, you can just assign a new value to an existing key:

>>> MLB\_team['Seattle'] = 'Seahawks'

Accessing Dictionary Values=============

A value is retrieved from a dictionary by specifying its corresponding key in square brackets ([]):

>>> MLB\_team['Minnesota']

'Twins'

>>> MLB\_team['Colorado']

'Rockies'

To delete an entry, use the del statement, specifying the key to delete:

>>> del MLB\_team['Seattle']

Building a Dictionary Incrementally=================

>>> person = {}

>>> person['fname'] = 'Joe'

>>> person['lname'] = 'Fonebone'

>>> person['age'] = 51

>>> person['spouse'] = 'Edna'

>>> person['children'] = ['Ralph', 'Betty', 'Joey']

>>> person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

>>> person

{'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',

'children': ['Ralph', 'Betty', 'Joey'], 'pets': {'dog': 'Fido', 'cat': 'Sox'}}

>>> 'Milwaukee' in MLB\_team

True

>>> 'Toronto' in MLB\_team

False

>>> 'Toronto' not in MLB\_team

True

Built-in Dictionary Methods=================

d.clear() empties dictionary d of all key-value pairs

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d

{'a': 10, 'b': 20, 'c': 30}

>>> d.clear()

>>> d

{}

d.get(<key>) searches dictionary d for <key> and returns the associated value if it is found. If <key> is not found, it returns None:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> print(d.get('b'))

20

>>> print(d.get('z'))

None

d.items() returns a list of tuples containing the key-value pairs in d. The first item in each tuple is the key, and the second item is the key’s value:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d

{'a': 10, 'b': 20, 'c': 30}

>>> list(d.items())

[('a', 10), ('b', 20), ('c', 30)]

d.keys() returns a list of all keys in d:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d

{'a': 10, 'b': 20, 'c': 30}

>>> list(d.keys())

['a', 'b', 'c']

d.values() returns a list of all values in d:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d

{'a': 10, 'b': 20, 'c': 30}

>>> list(d.values())

[10, 20, 30]

==Removes a key from a dictionary, if it is present, and returns its value.

If <key> is present in d, d.pop(<key>) removes <key> and returns its associated value:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.pop('b')

20

>>> d

{'a': 10, 'c': 30}

====Removes a key-value pair from a dictionary.

d.popitem() removes the last key-value pair added from d and returns it as a tuple:

>>> d = {'a': 10, 'b': 20, 'c': 30}

>>> d.popitem()

('c', 30)

>>> d

{'a': 10, 'b': 20}

The list class is a fundamental built-in data type in Python.

It has an impressive and useful set of features, allowing you to efficiently organize and manipulate heterogeneous data.

Properties:

Ordered: They contain elements or items that are sequentially arranged according to their specific insertion order.

Zero-based: They allow you to access their elements by indices that start from zero.

Mutable: They support in-place mutations or changes to their contained elements.

Heterogeneous: They can store objects of different types.

Growable and dynamic: They can grow or shrink dynamically, which means that they support the addition, insertion, and removal of elements.

Nestable: They can contain other lists, so you can have lists of lists.

Iterable: They support iteration, so you can traverse them using a loop or comprehension while you perform operations on each of their elements.

Sliceable: They support slicing operations, meaning that you can extract a series of elements from them.

Combinable: They support concatenation operations, so you can combine two or more lists using the concatenation operators.

Copyable: They allow you to make copies of their content using various techniques.

>>> colors = [

... "red",

... "orange",

... "yellow",

... "green",

... "blue",

... "indigo",

... "violet"

... ]

>>> colors

['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

Positions are numbered from zero to the length of the list minus one. The element at index 0 is the first element in the list, the element at index 1 is the second, and so on.

ex: colors[0]

Constructing Lists in Python

create lists using one of the following tools:

a. List literals

b. The list() constructor

c. A list comprehension

Ex:

digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list((0, 1, 2, 3, 4, 5, 6, 7, 8, 9))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list({"circle", "square", "triangle", "rectangle", "pentagon"})

['square', 'rectangle', 'triangle', 'pentagon', 'circle']

>>> list({"name": "John", "age": 30, "city": "New York"}.items())

[('name', 'John'), ('age', 30), ('city', 'New York')]

>>> list("Pythonista")

['P', 'y', 't', 'h', 'o', 'n', 'i', 's', 't', 'a']

>>> list()

[]

Building Lists With List Comprehensions :

>>> [number \*\* 2 for number in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Retrieving Multiple Items From a List: Slicing=======

list\_object[start:stop:step]

>>> letters = ["A", "a", "B", "b", "C", "c", "D", "d"]

>>> upper\_letters = letters[0::2] # Or [::2]

>>> upper\_letters

['A', 'B', 'C', 'D']

>>> lower\_letters = letters[1::2]

>>> lower\_letters

['a', 'b', 'c', 'd']

Copy a list:

>>> countries = ["United States", "Canada", "Poland", "Germany", "Austria"]

>>> nations = countries

\*\*If we make any change on countries, it will effect on nations also

>>> countries = ["United States", "Canada", "Poland", "Germany", "Austria"]

>>> nations = countries.copy()

\*\* it makes a shallow copy which a different list

Updating Items in Lists: Index Assignments

=================

To change the value of a given element in a list, you can use the following syntax:

list\_object[index] = new\_value

The .append() method is probably the most common tool that you’ll use to add items to an existing list. As its name suggests, this method allows you to append items to a list. The method takes one item at a time and adds it to the right end of the target list.

Extending a List With Multiple Items at Once: .extend()

>>> fruits = ["apple", "pear", "peach"]

>>> fruits.extend(["orange", "mango", "banana"])

>>> fruits

['apple', 'pear', 'peach', 'orange', 'mango', 'banana']

Inserting an Item at a Given Position: .insert()

>>> letters = ["A", "B", "F", "G"]

>>> letters.insert(2, "C")

>>> letters

['A', 'B', 'C', 'F', 'G']

>>> letters.insert(3, "D")

>>> letters

['A', 'B', 'C', 'D', 'F', 'G']

>>> letters.insert(4, "E")

>>> letters

['A', 'B', 'C', 'D', 'E', 'F', 'G']

Deleting Items From a List

.remove(item)

Removes the first occurrence of item from the list. It raises a ValueError if there’s no such item.

.pop([index])

Removes the item at index and returns it back to the caller. If you don’t provide a target index, then .pop() removes and returns the last item in the list. Note that the square brackets around index mean that the argument is optional. The brackets aren’t part of the syntax.

.clear()

Removes all items from the list.

>>> sample = [12, 11, 10, 42, 14, 12, 42]

>>> sample.remove(42)

>>> sample

[12, 11, 10, 14, 12, 42]

>>> sample.remove(42)

>>> sample

[12, 11, 10, 14, 12]

>>> sample.remove(42)

Traceback (most recent call last):

...

ValueError: list.remove(x): x not in list

>>> to\_visit = [

... "https://realpython.com",

... "https://python.org",

... "https://stackoverflow.com",

... ]

>>> visited = to\_visit.pop()

>>> visited

'https://stackoverflow.com'

>>> to\_visit

['https://realpython.com', 'https://python.org']

>>> visited = to\_visit.pop(0)

>>> visited

'https://realpython.com'

>>> to\_visit

['https://python.org']

>>> visited = to\_visit.pop(-1)

>>> visited

'https://python.org'

>>> to\_visit

[]

>>> cache = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> cache.clear()

>>> cache

[]

>>> colors = [

... "red",

... "orange",

... "yellow",

... "green",

... "blue",

... "indigo",

... "violet"

... ]

>>> del colors[1]

>>> colors

['red', 'yellow', 'green', 'blue', 'indigo', 'violet']

>>> del colors[-1]

>>> colors

['red', 'yellow', 'green', 'blue', 'indigo']

>>> del colors[2:4]

>>> colors

['red', 'yellow', 'indigo']

Reversing a List: reversed() and .reverse()

>>> digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> reversed(digits)

<list\_reverseiterator object at 0x10b261a50>

>>> list(reversed(digits))

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> digits

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> digits.reverse()

>>> digits

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> digits[::-1]

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Sorting a List: sorted() and .sort()

>>> numbers = [2, 9, 5, 1, 6]

>>> sorted(numbers)

[1, 2, 5, 6, 9]

>>> numbers

[2, 9, 5, 1, 6]

>>> words = ["Hello,", "World!", "I", "am", "a", "Pythonista!"]

>>> sorted(words)

['Hello,', 'I', 'Pythonista!', 'World!', 'a', 'am']

>>> numbers = [2, 9, 5, 1, 6]

>>> sorted(numbers, reverse=True)

[9, 6, 5, 2, 1]

-----------------------------

>>> employees = [

... ("John", 30, "Designer", 75000),

... ("Jane", 28, "Engineer", 60000),

... ("Bob", 35, "Analyst", 50000),

... ("Mary", 25, "Service", 40000),

... ("Tom", 40, "Director", 90000)

... ]

>>> sorted(employees, key=lambda employee: employee[1])

[

('Mary', 25, 'Service', 40000),

('Jane', 28, 'Engineer', 60000),

('John', 30, 'Designer', 75000),

('Bob', 35, 'Analyst', 50000),

('Tom', 40, 'Director', 90000)

]

In this example, you pass a lambda function to the key argument. This lambda takes an employee tuple as an argument and returns the age value, which lives at index 1. Then sorted() uses this value to sort the tuples.

>>> numbers = [2, 9, 5, 1, 6]

>>> numbers.sort()

>>> numbers

[1, 2, 5, 6, 9]

\*\*\*\*\*\*\*\*\*\*\*\*\*The main difference between sorted() and .sort() is that the former returns a new list of sorted data, while the latter sorts the target list in place. Also, because .sort() is a method, you need to call it on a list object.

------

Using a for Loop to Iterate Over a List

>>> colors = [

... "red",

... "orange",

... "yellow",

... "green",

... "blue",

... "indigo",

... "violet"

... ]

>>> for color in colors:

... print(color)

>>> for i in range(len(colors)):

... print(colors[i])

---

>>> integers = [1, 2, 3]

>>> letters = ["a", "b", "c"]

>>> floats = [4.0, 5.0, 6.0]

>>> for i, l, f in zip(integers, letters, floats):

... print(i, l, f)

...

1 a 4.0

2 b 5.0

3 c 6.0

==============Finding Items in a List

item in list\_object

item not in list\_object

Ex:

>>> usernames = ["john", "jane", "bob", "david", "eve"]

>>> "linda" in usernames

False

>>> "linda" not in usernames

True

>>> "bob" in usernames

True

>>> "bob" not in usernames

False

The .index() method is another tool that you can use to find a given value in an existing list. This method traverses a list looking for a specified value. If the value is in the list, then the method returns its index. Otherwise, it raises a ValueError exception:

>>> usernames = ["john", "jane", "bob", "david", "eve"]

>>> usernames.index("eve")

4

>>> usernames.index("linda")

Traceback (most recent call last):

...

ValueError: 'linda' is not in list

Getting the Length, Maximum, and Minimum of a List

>>> grades = [80, 97, 86, 100, 98, 82]

>>> n = len(grades)

>>> sum(grades) / n

90.5

>>> min(grades)

80

>>> max(grades)

100

=============Comparing Lists

You can also face the need to compare lists. Fortunately, list objects support the standard comparison operators. All these operators work by making item-by-item comparisons within the two involved lists:

>>> [2, 3] == [2, 3]

True

>>> [5, 6] != [5, 6]

False

>>> [5, 6, 7] < [7, 5, 6]

True

>>> [5, 6, 7] > [7, 5, 6]

False

>>> [4, 3, 2] <= [4, 3, 2]

True

>>> [4, 3, 2] >= [4, 3, 2]

True

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

python arrays are a data structure like lists. They contain a number of objects that can be of different data types. In addition, Python arrays can be iterated and have a number of built-in functions to handle them.

Python Array Syntax

Here is the basic syntax to declare an array in Python:

students = ['Alex', 'Bill', 'Catherine', 'Andy', 'Molly', 'Rose']

print(students)

each item has its own index number, we can access each item in our array. We can do so by calling our list and specifying an index number, like so:

print(students[2])

Slicing Items in a Python Array

We can also slice our Python lists to get multiple values. Let’s say we wanted to get the names in the middle of our list of students. By using a slice operation, we can retrieve multiple values in the list by their index number. These values will be retrieved by creating a range of index numbers, separating them using a colon.

Here’s an example of a slice in Python:

print(students[1:5])

Python Array Append and Pop

students = ['Alex', 'Bill', 'Catherine', 'Andy', 'Molly', 'Rose']

students.append('John')

print(students)

students = ['Alex', 'Bill', 'Catherine', 'Andy', 'Molly', 'Rose']

students.pop(0)

print(students)

Python Array Methods

=================

append() Adds an item to an array

pop() Removes an item from an array

clear() Removes all items from an array

copy() Returns a copy of an array

count() Returns the number of elements in a list

index() Returns the index of the first element with a specific value

insert() Adds an element to the array at a specific position

reverse() Reverses the order of the array

sort() Sorts the list

Array can be handled in Python by a module named array. They can be useful when we have to manipulate only a specific data type values.

A user can treat lists as arrays. However, user cannot constraint the type of elements stored in a list.

If you create arrays using the array module, all elements of the array must be of the same type.

# importing "array" for array creations

import array as arr

# creating an array with integer type

a = arr.array('i', [1, 2, 3])

# printing original array

print ("The new created array is : ", end =" ")

for i in range (0, 3):

print (a[i], end =" ")

print()

# creating an array with float type

b = arr.array('d', [2.5, 3.2, 3.3])

# printing original array

print ("The new created array is : ", end =" ")

for i in range (0, 3):

print (b[i], end =" ")

======================

Traverse an Array

You can traverse a Python array by using loops, like this one:

import array

balance = array.array('i', [300,200,100])

for x in balance:

print(x)

========================

Adding Elements to a Array

Elements can be added to the Array by using built-in insert() function.

Insert is used to insert one or more data elements into an array.

Based on the requirement, a new element can be added at the beginning, end, or any given index of array. append() is also used to add the value mentioned in its arguments at the end of the array.

# importing "array" for array creations

import array as arr

# array with int type

a = arr.array('i', [1, 2, 3])

print ("Array before insertion : ", end =" ")

for i in range (0, 3):

print (a[i], end =" ")

print()

# inserting array using

# insert() function

a.insert(1, 4)

print ("Array after insertion : ", end =" ")

for i in (a):

print (i, end =" ")

print()

# array with float type

b = arr.array('d', [2.5, 3.2, 3.3])

print ("Array before insertion : ", end =" ")

for i in range (0, 3):

print (b[i], end =" ")

print()

# adding an element using append()

b.append(4.4)

print ("Array after insertion : ", end =" ")

for i in (b):

print (i, end =" ")

print()

====================

Accessing elements from the Array

In order to access the array items refer to the index number.

Use the index operator [ ] to access an item in a array. The index must be an integer.

# importing array module

import array as arr

# array with int type

a = arr.array('i', [1, 2, 3, 4, 5, 6])

# accessing element of array

print("Access element is: ", a[0])

# accessing element of array

print("Access element is: ", a[3])

# array with float type

b = arr.array('d', [2.5, 3.2, 3.3])

# accessing element of array

print("Access element is: ", b[1])

# accessing element of array

print("Access element is: ", b[2])

=====================

Removing Elements from the Array

Elements can be removed from the array by using built-in remove() function but an Error arises if element doesn’t exist in the set. Remove() method only removes one element at a time, to remove range of elements, iterator is used.

pop() function can also be used to remove and return an element from the array, but by default it removes only the last element of the array, to remove element from a specific position of the array, index of the element is passed as an argument to the pop() method.

# importing "array" for array operations

import array

# initializing array with array values

# initializes array with signed integers

arr = array.array('i', [1, 2, 3, 1, 5])

# printing original array

print ("The new created array is : ", end ="")

for i in range (0, 5):

print (arr[i], end =" ")

print ("\r")

# using pop() to remove element at 2nd position

print ("The popped element is : ", end ="")

print (arr.pop(2))

# printing array after popping

print ("The array after popping is : ", end ="")

for i in range (0, 4):

print (arr[i], end =" ")

print("\r")

# using remove() to remove 1st occurrence of 1

arr.remove(1)

# printing array after removing

print ("The array after removing is : ", end ="")

for i in range (0, 3):

print (arr[i], end =" ")

================

Slicing of a Array

In Python array, there are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation.

Slice operation is performed on array with the use of colon(:).

# importing array module

import array as arr

# creating a list

l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

a = arr.array('i', l)

print("Initial Array: ")

for i in (a):

print(i, end =" ")

# Print elements of a range

# using Slice operation

Sliced\_array = a[3:8]

print("\nSlicing elements in a range 3-8: ")

print(Sliced\_array)

# Print elements from a

# pre-defined point to end

Sliced\_array = a[5:]

print("\nElements sliced from 5th "

"element till the end: ")

print(Sliced\_array)

# Printing elements from

# beginning till end

Sliced\_array = a[:]

print("\nPrinting all elements using slice operation: ")

print(Sliced\_array)

=======================

Searching element in a Array

In order to search an element in the array we use a python in-built index() method.

This function returns the index of the first occurrence of value mentioned in arguments.

# importing array module

import array

# initializing array with array values

# initializes array with signed integers

arr = array.array('i', [1, 2, 3, 1, 2, 5])

# printing original array

print ("The new created array is : ", end ="")

for i in range (0, 6):

print (arr[i], end =" ")

print ("\r")

# using index() to print index of 1st occurrenece of 2

print ("The index of 1st occurrence of 2 is : ", end ="")

print (arr.index(2))

# using index() to print index of 1st occurrenece of 1

print ("The index of 1st occurrence of 1 is : ", end ="")

print (arr.index(1))

================

Updating Elements in a Array

In order to update an element in the array we simply reassign a new value to the desired index we want to update.

# importing array module

import array

# initializing array with array values

# initializes array with signed integers

arr = array.array('i', [1, 2, 3, 1, 2, 5])

# printing original array

print ("Array before updation : ", end ="")

for i in range (0, 6):

print (arr[i], end =" ")

print ("\r")

# updating a element in a array

arr[2] = 6

print("Array after updation : ", end ="")

for i in range (0, 6):

print (arr[i], end =" ")

print()

# updating a element in a array

arr[4] = 8

print("Array after updation : ", end ="")

for i in range (0, 6):

print (arr[i], end =" ")

================

#function defined without any argument

#function does not return anything

def fun1():

print('God is good')

print('Goddess is good')

fun1()

fun1()

fun1()

======================

#function defined with an argument

#function does not return anything

def fun2(data):

print('Data is coming as ' , data)

fun2(12)

fun2('Sita')

fun2(23.45)

===================

#function defined with an argument

#function returns value

def fun3(data):

print('Data is coming as ' , data)

return data\*2

a = fun3(12)

print('Function return value is ' ,a)

===================

#function defined with multiple arguments, then arguments are separated using comma

#function returns a single value

def fun4(data1,data2):

print('Argument1 is coming as ' , data1)

print('Argument2 is coming as ' , data2)

return data1\*data2

a = fun4(12,3)

print('Function return value is ' ,a)

===================

#function defined with multiple arguments, then arguments are separated using comma

#function returns multiple values

def fun5(data1,data2):

print('Argument1 is coming as ' , data1)

print('Argument2 is coming as ' , data2)

return data1\*data2, data1-data2 , data1+data2, data1/data2

a,b,c,d= fun5(12,3)

print('Multiply ' ,a)

print('Substraction ' ,b)

print('Sum ' ,c)

print('Division ' ,d)

================

A function has argument, so when you called, you need to provide its value

But if we forget to give this argument value, its an error

If we need to give its value by default , then you have to declare it as :

def fun6(data1=2,data2=1)

Then if we forget to give its argument values, then default values are taken.

But remember: Values are assigned always Left To Right , means first 'data1' , then 'data2' then........

def fun6(data1=2,data2=1):

print('Argument1 is coming as ' , data1)

print('Argument2 is coming as ' , data2)

return data1\*data2

a= fun6(12,5)

print('Result ' ,a)

a= fun6(12)

print('Result ' ,a)

a= fun6()

print('Result ' ,a)

==============================

Design a function to check a given number is prime or not ?

def primeCheck(number):

flag=True

for i in range(2, int(number/2)+1):

if number%i==0:

flag=False

break;

return flag

n = int(input('Enter the number:'))

if( primeCheck(n)==True):

print(n , ' is a prime number')

else:

print(n , ' is not a prime number')

===========

Design a function to get all prime numbers between 1 to 100

def primeCheck(number):

flag=True

for i in range(2, int(number/2)+1):

if number%i==0:

flag=False

break;

return flag

for i in range(2,101):

if primeCheck(i)==True :

print(i , end=' ')

===========

Design a function to get first 5 prime numbers

def primeCheck(number):

flag=True

for i in range(2, int(number/2)+1):

if number%i==0:

flag=False

break;

return flag

count=0

n=2

while count<5:

if primeCheck(n)==True :

print(n , end=' ')

count = count+1

n = n+1

===============

There is a list of numbers. Show all prime numbers present in this list

def primeCheck(number):

flag=True

for i in range(2, int(number/2)+1):

if number%i==0:

flag=False

break;

return flag

mylist = [12,11,23,17,18,26,29]

for i in mylist:

if primeCheck(i)==True:

print(i, end=' ')

========

Design a number is an armstrong number or not

153 = 1\*1\*1 + 5\*5\*5+ 3\*3\*3

Design all 3 digit armstrong numbers

Design a function where you pass a list of numbers and find sum and avg

Design a function which is used to check a number is palindrome or not

121 == palindrome number

There is a list of numbers, find out what numbers in them are palindrome numbers among them.

design a function to check whether a given number is a perfect number or not

6 = 1+2+3 => is a perfect number

28 = 1+2+4+7+14 => is a perfect number

There is a list of numbers, find out what numbers in them are perfect numbers among them.

Python MySQL Tutorial

Introduction to MySQL Python connector

To access the MySQL database from Python, you need a database driver. MySQL Connector/Python is a standardized database driver provided by MySQL.

MySQL Connector/Python is designed specifically to MySQL. It supports all MySQL extensions such as LIMIT clause.

Installation

The MySQL Python Connector is available on pypi.org, therefore, you can install it using the pip command.

The pip command allows you to install MySQL Python connector on any Operating system including Windows, macOS, Linux, and Unix:

pip install mysql-connector-python

If you found any issue while installing, you can explicitly specify the module version as follows:

pip install mysql-connector-python==8.0.17

Note that to uninstall current MySQL Connector/Python, you use the following command:

pip uninstall mysql-connector-python

To verify the installation, you use the following steps:

Open Python command line

Type the following code

>>> import mysql.connector

>>> mysql.connector.connect(host='localhost',database='mysql',user='root',password='your pass')

If you see the following output, it means that you have been successfully installing the MySQL Connector/Python on your system.

<mysql.connector.connection.MySQLConnection object at 0x0187AE50>

Connect to a MySQL Database in Python:

import mysql.connector

from mysql.connector import Error

def connect():

""" Connect to MySQL database """

conn = None

try:

conn = mysql.connector.connect(host='localhost',

database='python\_mysql',

user='root',

password='SecurePass1!')

if conn.is\_connected():

print('Connected to MySQL database')

except Error as e:

print(e)

finally:

if conn is not None and conn.is\_connected():

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

connect()

Query Data from a Table in Python

To query data in a MySQL database from Python, you need to do the following steps:

Connect to the MySQL Database, you get a MySQLConnection object.

Instantiate a MySQLCursor object from the the MySQLConnection object.

Use the cursor to execute a query by calling its execute() method.

Use fetchone() , fetchmany() or fetchall() method to fetch data from the result set.

Close the cursor as well as the database connection by calling the close() method of the corresponding object.

Querying data with fetchone() method:

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def query\_with\_fetchone():

try:

dbconfig = read\_db\_config()

conn = MySQLConnection(\*\*dbconfig)

cursor = conn.cursor()

cursor.execute("SELECT \* FROM books")

row = cursor.fetchone()

while row is not None:

print(row)

row = cursor.fetchone()

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

query\_with\_fetchone()

Querying data with fetchall() method

In case the number of rows in the table is small, you can use the fetchall() method to fetch all rows from the database table. See the following code.

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def query\_with\_fetchall():

try:

dbconfig = read\_db\_config()

conn = MySQLConnection(\*\*dbconfig)

cursor = conn.cursor()

cursor.execute("SELECT \* FROM books")

rows = cursor.fetchall()

print('Total Row(s):', cursor.rowcount)

for row in rows:

print(row)

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

query\_with\_fetchall()

Code language: Python (python)

Connecting to MySQL Database using MySQLConnection object

First, create a database configuration file named config.ini and define a section with four parameters as follows:

[mysql]

host = localhost

database = python\_mysql

user = root

password =SecurePass1!

Second, create a new module named python\_mysql\_dbconfig.py that reads the database configuration from the config.ini file and returns a dictionary object:

from configparser import ConfigParser

def read\_db\_config(filename='config.ini', section='mysql'):

""" Read database configuration file and return a dictionary object

:param filename: name of the configuration file

:param section: section of database configuration

:return: a dictionary of database parameters

"""

# create parser and read ini configuration file

parser = ConfigParser()

parser.read(filename)

# get section, default to mysql

db = {}

if parser.has\_section(section):

items = parser.items(section)

for item in items:

db[item[0]] = item[1]

else:

raise Exception('{0} not found in the {1} file'.format(section, filename))

return db

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def connect():

""" Connect to MySQL database """

db\_config = read\_db\_config()

conn = None

try:

print('Connecting to MySQL database...')

conn = MySQLConnection(\*\*db\_config)

if conn.is\_connected():

print('Connection established.')

else:

print('Connection failed.')

except Error as error:

print(error)

finally:

if conn is not None and conn.is\_connected():

conn.close()

print('Connection closed.')

if \_\_name\_\_ == '\_\_main\_\_':

connect()

Querying data with fetchmany() method

For a relatively big table, it takes time to fetch all rows and return the entire result set. In addition, fetchall() needs to allocate enough memory to store the entire result set in the memory, which is not efficient.

MySQL Connector/Python has the fetchmany() method that returns the next number of rows (n) of the result set, which allows you to balance between retrieval time and memory space.

def iter\_row(cursor, size=10):

while True:

rows = cursor.fetchmany(size)

if not rows:

break

for row in rows:

yield row

def query\_with\_fetchmany():

try:

dbconfig = read\_db\_config()

conn = MySQLConnection(\*\*dbconfig)

cursor = conn.cursor()

cursor.execute("SELECT \* FROM books")

for row in iter\_row(cursor, 10):

print(row)

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

Insert Data Into a Table

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def insert\_book(title, isbn):

query = "INSERT INTO books(title,isbn) " \

"VALUES(%s,%s)"

args = (title, isbn)

try:

db\_config = read\_db\_config()

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

cursor.execute(query, args)

if cursor.lastrowid:

print('last insert id', cursor.lastrowid)

else:

print('last insert id not found')

conn.commit()

except Error as error:

print(error)

finally:

cursor.close()

conn.close()

def main():

insert\_book('A Sudden Light','9781439187036')

if \_\_name\_\_ == '\_\_main\_\_':

main()

Insert multiple rows into a table

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def insert\_books(books):

query = "INSERT INTO books(title,isbn) " \

"VALUES(%s,%s)"

try:

db\_config = read\_db\_config()

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

cursor.executemany(query, books)

conn.commit()

except Error as e:

print('Error:', e)

finally:

cursor.close()

conn.close()

def main():

books = [('Harry Potter And The Order Of The Phoenix', '9780439358071'),

('Gone with the Wind', '9780446675536'),

('Pride and Prejudice (Modern Library Classics)', '9780679783268')]

insert\_books(books)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Update Data in a Table

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def update\_book(book\_id, title):

# read database configuration

db\_config = read\_db\_config()

# prepare query and data

query = """ UPDATE books

SET title = %s

WHERE id = %s """

data = (title, book\_id)

try:

conn = MySQLConnection(\*\*db\_config)

# update book title

cursor = conn.cursor()

cursor.execute(query, data)

# accept the changes

conn.commit()

except Error as error:

print(error)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

update\_book(37, 'The Giant on the Hill \*\*\* TEST \*\*\*')

Delete Data from a Table

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def delete\_book(book\_id):

db\_config = read\_db\_config()

query = "DELETE FROM books WHERE id = %s"

try:

# connect to the database server

conn = MySQLConnection(\*\*db\_config)

# execute the query

cursor = conn.cursor()

cursor.execute(query, (book\_id,))

# accept the change

conn.commit()

except Error as error:

print(error)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

delete\_book(102)

Call Stored Procedures in Python

Make procedures in MySQL side:

USE python\_mysql;

DELIMITER $$

CREATE PROCEDURE find\_all()

BEGIN

SELECT

title,

isbn,

CONCAT(first\_name,' ',last\_name) AS author

FROM books

INNER JOIN book\_author

ON book\_author.book\_id = books.id

INNER JOIN authors

ON book\_author.author\_id = authors.id

ORDER BY title;

END$$

DELIMITER ;

Calling the find\_all() stored procedure, you will get the following result set:

CALL find\_all();

Another procedure:

DELIMITER $$

CREATE PROCEDURE find\_by\_isbn(

IN p\_isbn VARCHAR(13),

OUT p\_title VARCHAR(255)

)

BEGIN

SELECT title

INTO p\_title

FROM books

WHERE isbn = p\_isbn;

END$$

DELIMITER ;

Execute By:

CALL find\_by\_isbn('1235927658929',@title);

SELECT @title;

Calling stored procedures from Python

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def call\_find\_all\_sp():

try:

db\_config = read\_db\_config()

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

cursor.callproc('find\_all')

# print out the result

for result in cursor.stored\_results():

print(result.fetchall())

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

call\_find\_all\_sp()

Another one:

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def call\_find\_by\_isbn():

try:

db\_config = read\_db\_config()

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

args = ['1236400967773', 0]

result\_args = cursor.callproc('find\_by\_isbn', args)

print(result\_args[1])

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

call\_find\_by\_isbn()

Read & Update BLOB in MySQL Database

Updating BLOB data in Python:::::

def read\_file(filename):

with open(filename, 'rb') as f:

photo = f.read()

return photo

from mysql.connector import MySQLConnection, Error

from python\_mysql\_dbconfig import read\_db\_config

def update\_blob(author\_id, filename):

# read file

data = read\_file(filename)

# prepare update query and data

query = "UPDATE authors " \

"SET photo = %s " \

"WHERE id = %s"

args = (data, author\_id)

db\_config = read\_db\_config()

try:

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

cursor.execute(query, args)

conn.commit()

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

def main():

update\_blob(144, "pictures\garth\_stein.jpg")

if \_\_name\_\_ == '\_\_main\_\_':

main()

Check From MySQL: SELECT \* FROM authors

WHERE id = 144;

Reading BLOB data in Python:::::::::::::::::::;;

def write\_file(data, filename):

with open(filename, 'wb') as f:

f.write(data)

def read\_blob(author\_id, filename):

# select photo column of a specific author

query = "SELECT photo FROM authors WHERE id = %s"

# read database configuration

db\_config = read\_db\_config()

try:

# query blob data form the authors table

conn = MySQLConnection(\*\*db\_config)

cursor = conn.cursor()

cursor.execute(query, (author\_id,))

photo = cursor.fetchone()[0]

# write blob data into a file

write\_file(photo, filename)

except Error as e:

print(e)

finally:

cursor.close()

conn.close()

def main():

read\_blob(144,"output\garth\_stein.jpg")

if \_\_name\_\_ == '\_\_main\_\_':

main()

Single inheritance

=============

A class can reuse another class by inheriting it.

When a child class inherits from a parent class, the child class can access the attributes and methods of the parent class.

For example, you can define an Employee class that inherits from the Person class:

class Employee(Person):

def \_\_init\_\_(self, name, age, job\_title):

super().\_\_init\_\_(name, age)

self.job\_title = job\_title

Inside the \_\_init\_\_ method of the Employee class calls the \_\_init\_\_method of the Person class to initialize the name and age attributes.

The super() allows a child class to access a method of the parent class.

The Employee class extends the Person class by adding one more attribute called job\_title.

The Person is the parent class while the Employee is a child class. To override the greet() method in the Person class, you can define the greet() method in the Employee class as follows:

class Employee(Person):

def \_\_init\_\_(self, name, age, job\_title):

super().\_\_init\_\_(name, age)

self.job\_title = job\_title

def greet(self):

return super().greet() + f" I'm a {self.job\_title}."

The greet() method in the Employee is also called the greet() method of the Person class. In other words, it delegates to a method of the parent class.

he following creates a new instance of the Employee class and call the greet() method:

employee = Employee('John', 25, 'Python Developer')

print(employee.greet())

Output:

Hi, it's John. I'm a Python Developer.

Introduction to Python overridding method

==========================================

The overriding method allows a child class to provide a specific implementation of a method that is already provided by one of its parent classes.

First, define the Employee class:

class Employee:

def \_\_init\_\_(self, name, base\_pay):

self.name = name

self.base\_pay = base\_pay

def get\_pay(self):

return self.base\_pay

The Employee class has two instance variables name and base\_pay. It also has the get\_pay() method that returns the base\_pay.

Second, define the SalesEmployee that inherits from the Employee class:

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

The SalesEmployee class has three instance attributes: name, base\_pay, and sales\_incentive.

Third, create a new instance of the SalesEmployee class and display the pay:

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

Output:

5000

The get\_pay() method returns only the base\_pay, not the sum of the base\_pay and sales\_incentive.

When you call the get\_pay() from the instance of the SalesEmployee class, Python executes the get\_pay() method of the Employee class, which returns the base\_pay.

To include the sales incentive in the pay, you need to redefine the get\_pay() method in the SalesEmployee class as follows:

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

def get\_pay(self):

return self.base\_pay + self.sales\_incentive

In this case, we say that the get\_pay() method in the SalesEmployee class overrides the get\_pay() method in the Employee class.

When you call the get\_pay() method of the SalesEmployee‘s object, Python will call the get\_pay() method in the SalesEmployee class:

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

Output:

6500

If you create an instance of the Employee class, Python will call the get\_pay() method of the Employee class, not the get\_pay() method of the SalesEmployee class. For example:

jane = Employee('Jane', 5000)

print(jane.get\_pay())

FULL CODE:

class Employee:

def \_\_init\_\_(self, name, base\_pay):

self.name = name

self.base\_pay = base\_pay

def get\_pay(self):

return self.base\_pay

class SalesEmployee(Employee):

def \_\_init\_\_(self, name, base\_pay, sales\_incentive):

self.name = name

self.base\_pay = base\_pay

self.sales\_incentive = sales\_incentive

def get\_pay(self):

return self.base\_pay + self.sales\_incentive

if \_\_name\_\_ == '\_\_main\_\_':

john = SalesEmployee('John', 5000, 1500)

print(john.get\_pay())

jane = Employee('Jane', 5000)

print(jane.get\_pay())

Introduction to Python Abstract Classes

In object-oriented programming, an abstract class is a class that cannot be instantiated. However, you can create classes that inherit from an abstract class.

Typically, you use an abstract class to create a blueprint for other classes.

Similarly, an abstract method is an method without an implementation. An abstract class may or may not include abstract methods.

Python doesn’t directly support abstract classes. But it does offer a module that allows you to define abstract classes.

To define an abstract class, you use the abc (abstract base class) module.

The abc module provides you with the infrastructure for defining abstract base classes.

For example:

from abc import ABC

class AbstractClassName(ABC):

To define an abstract method, you use the @abstractmethod decorator:

from abc import ABC, abstractmethod

class AbstractClassName(ABC):

@abstractmethod

def abstract\_method\_name(self):

Python abstract class example

from abc import ABC, abstractmethod

class Employee(ABC):

def \_\_init\_\_(self, first\_name, last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

@property

def full\_name(self):

return f"{self.first\_name} {self.last\_name}"

@abstractmethod

def get\_salary(self):

pass

class FulltimeEmployee(Employee):

def \_\_init\_\_(self, first\_name, last\_name, salary):

super().\_\_init\_\_(first\_name, last\_name)

self.salary = salary

def get\_salary(self):

return self.salary

class HourlyEmployee(Employee):

def \_\_init\_\_(self, first\_name, last\_name, worked\_hours, rate):

super().\_\_init\_\_(first\_name, last\_name)

self.worked\_hours = worked\_hours

self.rate = rate

def get\_salary(self):

return self.worked\_hours \* self.rate

class Payroll:

def \_\_init\_\_(self):

self.employee\_list = []

def add(self, employee):

self.employee\_list.append(employee)

def print(self):

for e in self.employee\_list:

print(f"{e.full\_name} \t ${e.get\_salary()}")

The main program

The following app.py uses the FulltimeEmployee, HourlyEmployee, and Payroll classes to print out the payroll of five employees.

from fulltimeemployee import FulltimeEmployee

from hourlyemployee import HourlyEmployee

from payroll import Payroll

payroll = Payroll()

payroll.add(FulltimeEmployee('John', 'Doe', 6000))

payroll.add(FulltimeEmployee('Jane', 'Doe', 6500))

payroll.add(HourlyEmployee('Jenifer', 'Smith', 200, 50))

payroll.add(HourlyEmployee('David', 'Wilson', 150, 100))

payroll.add(HourlyEmployee('Kevin', 'Miller', 100, 150))

payroll.print()

Output:

John Doe $6000

Jane Doe $6500

Jenifer Smith $10000

David Wilson $15000

Kevin Miller $15000

Introduction to the Python Multiple inheritance.

When a class inherits from a single class, you have single inheritance. Python allows a class to inherit from multiple classes. If a class inherits from two or more classes, you’ll have multiple inheritance.

To extend multiple classes, you specify the parent classes inside the parentheses () after the class name of the child class like this:

class ChildClass(ParentClass1, ParentClass2, ParentClass3):

class Car:

def go(self):

print('Going')

class Flyable:

def fly(self):

print('Flying')

class FlyingCar(Flyable, Car):

pass

Another example for multiple inheritence:

class Car:

def start(self):

print('Start the Car')

def go(self):

print('Going')

class Flyable:

def start(self):

print('Start the Flyable object')

def fly(self):

print('Flying')

class FlyingCar(Flyable, Car):

def start(self):

super().start()

if \_\_name\_\_ == '\_\_main\_\_':

car = FlyingCar()

car.start()

Output:

Start the Flyable object

Explanations:

As you can see clearly from the output, the super().start() calls the start() method of the Flyable class.

The following shows the \_\_mro\_\_ of the FlyingCar class:

print(FlyingCar.\_\_mro\_\_)

Code language: Python (python)

Output:

(<class '\_\_main\_\_.FlyingCar'>, <class '\_\_main\_\_.Flyable'>, <class '\_\_main\_\_.Car'>, <class 'object'>)

From left to right, you’ll see the FlyingCar, Flyable, Car, and object.

Note that the Car and Flyable objects inherit from the object class implicitly. When you call the start() method from the FlyingCar‘s object, Python uses the \_\_mro\_\_ class search path.

Since the Flyable class is next to the FlyingCar class, the super().start() calls the start() method of the FlyingCar class.

If you flip the order of Flyable and Car classes in the list, the \_\_mro\_\_ will change accordingly. For example:

# Car, Flyable classes...

class FlyingCar(Car, Flyable):

def start(self):

super().start()

if \_\_name\_\_ == '\_\_main\_\_':

car = FlyingCar()

car.start()

print(FlyingCar.\_\_mro\_\_)

Output:

Start the Car

(<class '\_\_main\_\_.FlyingCar'>, <class '\_\_main\_\_.Car'>, <class '\_\_main\_\_.Flyable'>, <class 'object'>)

In this example, the super().start() calls the start() method of the Car class instead, based on their orders in the method order resolution.

Introduction to Python exceptions

==========================================

In Python, exceptions are objects of the exception classes.

All exception classes are the subclasses of the BaseException class.

However, almost all built-in exception classes inherit from the Exception class, which is the subclass of the BaseException class:

The following example defines a list of three elements and attempts to access the fourth one:

colors = ['red', 'green', 'blue']

print(colors[3])

The invalid index caused the IndexError exception as expected:

IndexError: list index out of range

When an exception occurs, Python stops the program unless you handle it.

To handle an exception, you use the try...except statement. For example:

colors = ['red', 'green', 'blue']

try:

print(colors[3])

except IndexError as e:

print(e)

print('Continue to run')

Output:

<class 'IndexError'> - list index out of range

Continue to run

To handle exceptions, you use the try statement. The try statement has the following clauses:

try:

# code that you want to protect from exceptions

except <ExceptionType> as ex:

# code that handle the exception

finally:

# code that always execute whether the exception occurred or not

else:

# code that excutes if try execute normally (an except clause must be present)

try

In the try clause, you place the code that protects from one or more potential exceptions.

It’s a good practice to keep the code as short as possible. Often, you’ll have a single statement in the try clause.

The try clause appears exactly one time in the try statement.

except

In the except clause, you place the code that handles a specific exception type.

A try statement can have zero or more except clauses.

Typically, each except clause handles different exception types in specific ways.

In an except clause, the as ex is optional.

And the <ExceptionType> is also optional. However, if you omit the <ExceptionType> as ex, you’ll have a bare exception handler.

When specifying exception types in the except clauses, you place the most specific to least specific exceptions from top to bottom.

finally

The finally clause may appear zero or 1 time in a try statement. The finally clause always executes whether an exception occurred or not.

else

The else clause also appears zero or 1 time.

And the else clause is only valid if the try statement has at least one except clause.

Typically, you place the code that executes if the try clause terminates normally.

-----

If you have the same logic that handles different exception types, you can group them in a single except clause. For example:

try:

...

except <ExceptionType1> as ex:

log(ex)

except <ExceptionType2> as ex:

log(ex)

Become

try:

...

except (<ExceptionType1>, <ExceptionType2>) as ex:

log(ex)

It’s important to note that the except order matters because Python will run the first except clause whose exception type matches the occurred exception.

---------------------------

Another example:

def division(a, b):

try:

return {

'success': True,

'message': 'OK',

'result': a / b

}

except ZeroDivisionError as e:

return {

'success': False,

'message': 'b cannot be zero',

'result': None

}

result = division(10, 0)

print(result)

Now, if you don’t catch the ZeroDivisionError exception but the more general exception like Exception class:

def division(a, b):

try:

return {

'success': True,

'message': 'OK',

'result': a / b

}

except Exception as e:

return {

'success': False,

'message': 'b cannot be zero',

'result': None

}

result = division(10, 0)

print(result)

The program works as before because the try...except also catches the exception type that is the subclass of the Exception class.

However, if you pass two strings instead of two numbers to the division() function, you’ll get the same message as if the ZeroDivisionError exception occurred:

def division(a, b):

try:

return {

'success': True,

'message': 'OK',

'result': a / b

}

except Exception as e:

return {

'success': False,

'message': 'b cannot be zero',

'result': None

}

result = division('10', '2')

print(result)

Output:

{'success': False, 'message': 'b cannot be zero', 'result': None}

Therefore, you should always handle the exceptions from the most specific to the least specific.

For example:

def division(a, b):

try:

return {

'success': True,

'message': 'OK',

'result': a / b

}

except TypeError as e:

return {

'success': False,

'message': 'Both a & b must be numbers',

'result': None

}

except ZeroDivisionError as e:

return {

'success': False,

'message': 'b cannot be zero',

'result': None

}

except Exception as e:

return {

'success': False,

'message': str(e),

'result': None

}

result = division('10', '2')

print(result)

In this example, we catch the TypeError, ZeroDivisionError, and Exception in the order that they appear in the try...except statement.

If the code that handles different exceptions are the same, you can group all exceptions into one as follows:

def division(a, b):

try:

return {

'success': True,

'message': 'OK',

'result': a / b

}

except (TypeError, ZeroDivisionError, Exception) as e:

return {

'success': False,

'message': str(e),

'result': None

}

result = division(10, 0)

print(result)

Output:

{'success': False, 'message': 'division by zero', 'result': None}

A better approach is to raise an exception to the caller if the ZeroDivisionError exception occurred.

For example:

def divide(a, b):

try:

return a / b

except ZeroDivisionError as ex:

raise ValueError('The second argument (b) must not be zero')

try:

result = divide(10, 0)

except ValueError as e:

print(e)

else:

print('result:', result)

Output:

The second argument (b) must not be zero

===================================================================

To create a custom exception class, you define a class that inherits from the built-in Exception class or one of its subclasses such as ValueError class:

The following example defines a CustomException class that inherits from the Exception class:

class CustomException(Exception):

""" my custom exception class """

Note that the CustomException class has a docstring that behaves like a statement. Therefore, you don’t need to add the pass statement to make the syntax valid.

To raise the CustomException, you use the raise statement. For example, the following uses the raise statement to raise the CustomException:

class CustomException(Exception):

""" my custom exception class """

try:

raise CustomException('This is my custom exception')

except CustomException as ex:

print(ex)

Output:

This is my custom exception

Python custom exception example

class FahrenheitError(Exception):

min\_f = 32

max\_f = 212

def \_\_init\_\_(self, f, \*args):

super().\_\_init\_\_(args)

self.f = f

def \_\_str\_\_(self):

return f'The {self.f} is not in a valid range {self.min\_f, self.max\_f}'

How it works.

First, define the FahrenheitError class that inherits from the Exception class.

Second, add two class attributes min\_f and max\_f that represent the minimum and maximum Fahrenheit values.

Third, define the \_\_init\_\_ method that accepts a Fahrenheit value (f) and a number of position arguments (\*args). In the \_\_init\_\_ method, call the \_\_init\_\_ method of the base class. Also, assign the f argument to the f instance attribute.

Finally, override the \_\_str\_\_ method to return a custom string representation of the class instance.

Define the fahrenheit\_to\_celsius function

The following defines the fahrenheit\_to\_celsius function that accepts a temperature in Fahrenheit and returns a temperature in Celcius:

def fahrenheit\_to\_celsius(f: float) -> float:

if f < FahrenheitError.min\_f or f > FahrenheitError.max\_f:

raise FahrenheitError(f)

return (f - 32) \* 5 / 9

The fahrenheit\_to\_celsius function raises the FahrenheitError excpetion if the input temperature is not in the valid range. Otherwise, it converts the temperature from Fahrenheit to Celcius.

========================

Create the main program

The following main program uses the fahrenheit\_to\_celsius function and the FahrenheitError custom exception class:

if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter a temperature in Fahrenheit:')

try:

f = float(f)

except ValueError as ex:

print(ex)

else:

try:

c = fahrenheit\_to\_celsius(float(f))

except FahrenheitError as ex:

print(ex)

else:

print(f'{f} Fahrenheit = {c:.4f} Celsius')

Full Code:

class FahrenheitError(Exception):

min\_f = 32

max\_f = 212

def \_\_init\_\_(self, f, \*args):

super().\_\_init\_\_(args)

self.f = f

def \_\_str\_\_(self):

return f'The {self.f} is not in a valid range {self.min\_f, self.max\_f}'

def fahrenheit\_to\_celsius(f: float) -> float:

if f < FahrenheitError.min\_f or f > FahrenheitError.max\_f:

raise FahrenheitError(f)

return (f - 32) \* 5 / 9

if \_\_name\_\_ == '\_\_main\_\_':

f = input('Enter a temperature in Fahrenheit:')

try:

f = float(f)

except ValueError as ex:

print(ex)

else:

try:

c = fahrenheit\_to\_celsius(float(f))

except FahrenheitError as ex:

print(ex)

else:

print(f'{f} Fahrenheit = {c:.4f} Celsius')

ETL Pipeline with Python

ETL stands for Extract, Transform and Load. Data is often distributed across a variety of different applications and systems. A Data Warehouse would be required to bring all of these diverse Data Sources together in a digestible format to generate significant insights that can help in business development.

We will begin with a basic ETL Pipeline consisting of essential elements needed to extract the data, then transform it, and finally, load it into the right places. At this step, things are not as complex as they might seem, even if you are a complete beginner at it.

Importing the Right Packages

import requests

import pandas as pd

from sqlalchemy import create\_engine

Step 1: Extract

def extract()-> dict:

""" This API extracts data from

http://universities.hipolabs.com

"""

API\_URL = "http://universities.hipolabs.com/search?country=United+States"

data = requests.get(API\_URL).json()

return data

Step 2: Transform

def transform(data:dict) -> pd.DataFrame:

""" Transforms the dataset into desired structure and filters"""

df = pd.DataFrame(data)

print(f"Total Number of universities from API {len(data)}")

df = df[df["name"].str.contains("California")]

print(f"Number of universities in california {len(df)}")

df['domains'] = [','.join(map(str, l)) for l in df['domains']]

df['web\_pages'] = [','.join(map(str, l)) for l in df['web\_pages']]

df = df.reset\_index(drop=True)

return df[["domains","country","web\_pages","name"]]

Step 3: Load

def load(df:pd.DataFrame)-> None:

""" Loads data into a sqllite database"""

disk\_engine = create\_engine('sqlite:///my\_lite\_store.db')

df.to\_sql('cal\_uni', disk\_engine, if\_exists='replace')

Running your first ETL pipeline

data = extract()

df = transform(data)

load(df)

=========================================

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

Empty cells

Data in wrong format

Wrong data

Duplicates

Our Data Set

In the next chapters we will use this data set:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 2020/12/26 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains wrong format ("Date" in row 26).

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).

Pandas - Cleaning Empty Cells

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

import pandas as pd

df = pd.read\_csv('data.csv')

new\_df = df.dropna()

print(new\_df.to\_string())

Note: By default, the dropna() method returns a new DataFrame, and will not change the original.

If you want to change the original DataFrame, use the inplace = True argument:

import pandas as pd

df = pd.read\_csv('data.csv')

df.dropna(inplace = True)

print(df.to\_string())

Replace Empty Values

Another way of dealing with empty cells is to insert a new value instead.

Example

Replace NULL values with the number 130:

import pandas as pd

df = pd.read\_csv('data.csv')

df.fillna(130, inplace = True)

Replace Only For Specified Columns

Example

Replace NULL values in the "Calories" columns with the number 130:

import pandas as pd

df = pd.read\_csv('data.csv')

df["Calories"].fillna(130, inplace = True)

Cleaning Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Convert Into a Correct Format

In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaN 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 20201226 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a to\_datetime() method for this:

ExampleGet your own Python Server

Convert to date:

import pandas as pd

df = pd.read\_csv('data.csv')

df['Date'] = pd.to\_datetime(df['Date'])

print(df.to\_string())

Result:

Duration Date Pulse Maxpulse Calories

0 60 '2020/12/01' 110 130 409.1

1 60 '2020/12/02' 117 145 479.0

2 60 '2020/12/03' 103 135 340.0

3 45 '2020/12/04' 109 175 282.4

4 45 '2020/12/05' 117 148 406.0

5 60 '2020/12/06' 102 127 300.0

6 60 '2020/12/07' 110 136 374.0

7 450 '2020/12/08' 104 134 253.3

8 30 '2020/12/09' 109 133 195.1

9 60 '2020/12/10' 98 124 269.0

10 60 '2020/12/11' 103 147 329.3

11 60 '2020/12/12' 100 120 250.7

12 60 '2020/12/12' 100 120 250.7

13 60 '2020/12/13' 106 128 345.3

14 60 '2020/12/14' 104 132 379.3

15 60 '2020/12/15' 98 123 275.0

16 60 '2020/12/16' 98 120 215.2

17 60 '2020/12/17' 100 120 300.0

18 45 '2020/12/18' 90 112 NaN

19 60 '2020/12/19' 103 123 323.0

20 45 '2020/12/20' 97 125 243.0

21 60 '2020/12/21' 108 131 364.2

22 45 NaT 100 119 282.0

23 60 '2020/12/23' 130 101 300.0

24 45 '2020/12/24' 105 132 246.0

25 60 '2020/12/25' 102 126 334.5

26 60 '2020/12/26' 100 120 250.0

27 60 '2020/12/27' 92 118 241.0

28 60 '2020/12/28' 103 132 NaN

29 60 '2020/12/29' 100 132 280.0

30 60 '2020/12/30' 102 129 380.3

31 60 '2020/12/31' 92 115 243.0

As you can see from the result, the date in row 26 was fixed, but the empty date in row 22 got a NaT (Not a Time) value, in other words an empty value. One way to deal with empty values is simply removing the entire row.

Removing Rows

The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

Example

Remove rows with a NULL value in the "Date" column:

df.dropna(subset=['Date'], inplace = True)

Fixing Wrong Data

"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

Set "Duration" = 45 in row 7:

df.loc[7, 'Duration'] = 45

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

for x in df.index:

if df.loc[x, "Duration"] > 120:

df.loc[x, "Duration"] = 120

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

Example

Delete rows where "Duration" is higher than 120:

for x in df.index:

if df.loc[x, "Duration"] > 120:

df.drop(x, inplace = True)

Removing Duplicates

Duplicate rows are rows that have been registered more than one time.

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row:

Returns True for every row that is a duplicate, othwerwise False:

import pandas as pd

df = pd.read\_csv('data.csv')

print(df.duplicated())

To remove duplicates, use the drop\_duplicates() method.

Example

Remove all duplicates:

df.drop\_duplicates(inplace = True)

Remember: The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.

Another example:

How to perform Data Cleaning in Python?

The processing of missing data is one of the most important imperfections in a dataset. Several methods for dealing with missing data are provided by the pandas package in Python, including dropna() and fillna(). The dropna() method is used to eliminate any columns or rows that have missing values.

import pandas as pd

data = pd.read\_csv('data.csv')

data = data.dropna()

The fillna() function can be used to fill in missing values with a specific value or method. For example, the following code will fill in missing values in the 'age' column with the mean age of the data:

import pandas as pd

data = pd.read\_csv('data.csv')

data['age'].fillna(data['age'].mean(), inplace=True)

Handling Outliers

Handling outliers is a typical data cleaning activity. Values that diverge greatly from the rest of the data are considered outliers. These factors should be managed carefully since they have a significant influence on a model's performance. The RobustScaler class from the scikit-learn toolkit in Python is used to handle outliers. By deleting the median and scaling the data according to the interquartile range, this class may be used to scale the data.

from sklearn.preprocessing import RobustScaler

data = pd.read\_csv('data.csv')

scaler = RobustScaler()

data = scaler.fit\_transform(data)

Encoding Categorical Variables

Another common data cleaning task is converting data into a format that can be used by a model. For instance, before categorical data can be employed in a model, it must be transformed into numerical data. The get\_dummies() method in the pandas package allows one to transform category data into numerical data. In the example below, the categorical feature ‘Department’ is transformed into numerical data:

import pandas as pd

data = pd.read\_csv('data.csv')

data = pd.get\_dummies(data, columns=['Department'])

Removing Duplicate Data

Duplicate data must also be eliminated during the data cleaning process. To delete duplicate rows from a Python DataFrame, the drop\_duplicates() method provided by the pandas package can be used. For instance, the code below will eliminate any redundant rows from the data:

import pandas as pd

data = pd.read\_csv('data.csv')

data = data.drop\_duplicates()

Here are some things you can do with Pandas to automate the data-cleaning process:

-------------------------------------------------------------

Task Function Used Description

Remove duplicates drop\_duplicates() Remove duplicate rows from a dataframe.

Drop missing values dropna() Remove rows or columns with missing values.

Impute missing values fillna() Fill in missing values in a dataframe with a specified value or method.

Convert data types astype() Convert the data type of a column in a dataframe.

Rename columns rename() Rename columns in a dataframe.

Group and

aggregate data groupby(), agg(), apply() Group and aggregate data in a dataframe.

Filter

data query(), loc[], iloc[] Filter data in a dataframe using various methods

Apply functions

to data apply() Apply a function to a column or row in a dataframe

Merge data merge(), join(), concat() Merge data from multiple dataframes

Pivot data pivot\_table() The method allows for more advanced features such as multi-index and custom aggregation.

By using these functions and methods, you can create a powerful data-cleaning pipeline in Pandas to automate the data-cleaning process.

LABS

Check whether the given sentence is alliterative or not

is\_alliterative(sentence)

#write code here

This method takes the sentence as the argument.

If the no. of words in the sentence is less than 2 words, return False to the caller method.

If there are more than 2 words in the sentence, and yet they begin with vowels (including upper case), return False to the caller method.

If there are more than 2 words in the sentence, and yet they all begin with different consonants (non-alliterative), return False to the caller method.

If there are more than 2 or equal to 2 words in the sentence, and if they all begin with the same consonant, irrespective of the case(i.e., alliterative), return True to the caller method.

Sample Input 1:

Enter the sentence to be validated for alliteration: She sells sea shells

Sample Output 2:

The sentence is alliterative

Sample Input 2:

Enter the sentence to be validated for alliteration: Ann sells sea shells

Sample Output 2:

The sentence is not alliterative

===================== Done

Find the relationship between users

def find\_relationship(name1,name2)

#write code here

This method should take two names as arguments. Find the total length of the names, and get the reminder value by dividing the total length of the name by 6. Based on this reminder value, decide the relationship.

Refer to the table below for deciding the relationship:

Reminder Return value

0 Soulmates

1 Colleagues

2 Friends

3 Good friends

4 Best friends

5 Close friends

Once the relationship is identified, then return the relationship value to the caller method.

Note:To find out the length of the name, do not consider the space in the name.

Sample Input 1:

Enter the name 1:Glenn

Enter the name 2:Kim

Sample Output 1:

Friends

Sample Input 2:

Enter the name 1:Lilly

Enter the name 2:Lenny

Sample Output 2:

Best friends

================

Find the discount amount.

def calculate\_discount(input\_string):

#fill code here

This method takes a string as its argument.

Split the input string based on the colon (':') with colon-separated.

Find the sum of the digits in house number and calculate the discount amount based on the below conditions:

House Type House Cost House Number sum Discount (%)

2BHK 3900000 Odd Number 4

3BHK 5100000 Odd Number 8

2BHK 3700000 Even Number 5

3BHK 4900000 Even Number 7

Calculate the discount amount and return the same.

Sample Input 1:

Enter the details:

123:2BHK

Sample Output 1:

185000.0

Sample Input 2:

Enter the details:

435:3BHK

Sample Output 2:

343000.0

============================ Done-1

Top Tier Motors is one of the leading multi-branded showroom for two-wheelers. The company plans to promote sales and decide to give discounts on vehicles for the following brands.

Brand Name Discount Percentage

TVS 10

Honda 5

Yamaha 7

Sample Input and Output:

Enter the vehicle name and price of Vehicle :

TVSNTORQ:72000

Price after discount for TVSNTORQ is xxxxxxx.xx

Thank you for using the Application

============================================================== Done

Suppose we have a set of integers.

Take a number from user and check whether it is present or not in given set of values.

if not present , show "{Data} is absent"

if present , show "{data} is present {no\_of\_times} times only "

============================================================== Done

Take a number from user and check whether it is a fibonacci number or not .

Input-1:

Enter Number : 0

Output-1:

Fibonacci number

Input-2:

Enter Number : 1

Output-2:

Fibonacci number

Input-3:

Enter Number : 5

Output-3:

Fibonacci number

Input-4:

Enter Number : 6

Output-4:

Not a fibonacci number

=======================================Done

Input-1 : 2A3E

output-1 : AAEEE

def makeString(str):

#return string

================================================================Done

def removeSpace(str):

#return string where all spaces will be removed

def removeDuplicates(str):

#remove all spaces and duplicate chars from given string and then return it

================================================================Done

def removePunctuation(str):

#return string where all spaces and punctuations will be removed

def checkPalindrome(str):

#return true if str is palindrome otherwise false

#before checking str , remove all space and punctuations from string

================Done

#def isPerfect(n):

# check number is perfect or not

#def showAllPerfect(listNumbers):

# show all perfect numbers between the given list of numbers

# show at end total number of perfect also

# if there is no prime, show "there is no perfect numbers"

# use isPerfect() method from here

list=[12,6,34,28,61,71,99]

showAllPrimes(list)

================Done

#def isPrime(n):

# check number is prime or not

#def showAllPrimes(listNumbers):

# show all prime numbers between the given list of numbers

# show at end total number of prime also

# if there is no prime, show "there is no prime numbers"

# use isPrime() method from here

list=[12,11,34,44,61,71,99]

showAllPrimes(list)

=================================== Done

Find the lucky number from the input string.

def find\_lucky\_number(dob):

#code here

This method takes the date of birth (as a string) in the format dd/mm/yyyy as an argument.

Validate the string according to the following condition:

In the input string, the first two characters representing a day should be between 01 and 31, the next character should be a slash ('/'), and the following two characters representing a month should be between 01 and 12, the next character should be a slash ('/'), and the following character representing a year should be less than 2023. Eg:28/08/1999

If the string is valid, then add the day, month, and year. Then sum each digit of the added values and return the same.

If the input string is not valid, the function should return the message as 'Invalid format'.

For example: If the string (date of birth) entered is 28/08/1999, then the lucky number is calculated as,

28 + 08 + 1999= 2035

2+0+3+5=10

The lucky number is 10.

Note: Do not use date functions. Consider the entered date format value as a string and do the specified manipulations.

Sample Input 1:

Enter the date of birth

24/04/1990

Sample Output 1:

The lucky number is 11

Sample Input 2:

Enter the date of birth

11-11-2001

Sample Output 2:

Invalid format

=========

Generate the code for the product

def generate\_code(product\_details):

#code write here

This method takes product\_details as arguments.

Product details contain product name, destination, month, and year (productName:destination:month:year) as colon-separated values.

The function should split the product\_details based on the (':') colon separator and then validate the details.

The validation rules are:

The length of the product name and the length of the destination should be greater than 3. The month should be between 1 to 12 (inclusive), and the length of the year should be 4.

If all the details are valid, then generate the product code.

The product code format should be Product\_name/destination/month\_year.

The product\_name in the product code should be formed as:

1. If the length of the product name is an odd number, the product code will be the first 3 characters, For example, The product code for mango will be MAN

2. If the length of the product name is an even number, the code will be the last 3 characters. For example, the product code for grapes will be PES.

3. Generated product name should be in upper case.

The destination in the product code should be the first and the last characters of the destination in upper case.

For example, if the destination is Florida, then the destination in the product code should be: FA

The month with the year in the product code should be formed as the month followed by the last 2 digits in the year.

For example: if the month and year are 9 and 2019, then the month\_year should be '919'.

Example:

input:

Sanitizer: Florida:9:2019

Output:

SAN/FA/919

Note: Do not consider the space in the product name and destination.

If the entered product detail is not valid, then display the message: Invalid product details.

Sample Input 1:

Mask: Pune:11:2019

Sample Output 1:

ASK/PE/1119

Sample Input 2:

Sanitizer: Florida:9:2019

Sample Output 2:

SAN/FA/919

Sample Input 3:

Sanitizer: Bo:13:2019

Sample Output 3:

Invalid product details

=========================

def isPalindrome(str):

# if string has any punctuations(include space) , remove it first

# convert the full string in lowercase

# now check palindrome testing , return true if str is palindrome otherwise false

def countPalindrome(listWords):

# return total count value for those words in listWords who are palindrome in nature

===============================================Done

Suppose we have a set of integers.

Take a number from user and check whether it is present or not in given set of values.

if not present , show "{Data} is absent"

if present , show "{data} is present {no\_of\_times} times only "

def checkOccurence(listNumbers, data):

# search data in listNumbers

# if present return msg as "{data} is present {no\_of\_times} times only "

# if not present return msg as "{Data} is absent"

=============================================== Done

Problem: Take a number for user and show its all possible factors in this manner:

Input : 12

Output:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 6

def makePyramid(n) :

#write code here

============================

The game is about one player saying a sentence and the other player has to arrange the letters in that word in alphabetical order.

Requirements:

Assume the letters in the sentence is in lowercase

The sentence should contain only alphabets and space, else print "<sentence> is an invalid input"

Sample Input/Output 1:

Enter the sentence

the series

Output:eht eeirss

Sample Input/Output 2:

Enter the sentence

5 is greater than 2

Output:5 is greater than 2 is an invalid input

================================================================

=================================================================

====================================assignments on data structure

Problem:

There is a list of numbers in a list as :

A=[12,23,12,23,11,12,24,-12,12]

Find each number and its occurence like:

{12: 4, 23:2 , 11: 1, 24:1 }

Ignore all negative and zero values.

Problem :

Take a string from user.If any space coming within string, plz ignore that.

Check all characters should be lower case letter only, otherwise show message "Invalid input".

Now need to show each character and its freequency in respect of a dictionary.

Like:

Sample Input:

Enter the word:

dibak ar

Output:

{ 'd':1, 'i':1, 'b':1, 'a':2 , 'k':1, 'r':1 }

Sample Input:

Enter the word:

dibak ar12

Output:

Invalid input

Problem :

If the no. of meanings entered is zero or negative, then display the message "Invalid Input" and display the contents of the dictionary ( key-value pairs) currently present (if any).

When you want to add more elements to the dictionary, you have to enter '1' else you need to enter '0'.

If the choice is not '0' or '1', then display the message "Invalid Input" followed by the contents of the list of dictionaries.

Sample Input :

Enter the word: Python

Enter the no of meanings: 2

Enter the meanings :

A large heavy-bodied snake

A high-level programming language

Do you want to add one more element to the dictionary? If yes, press 1, else press 0: 1

Enter the word: ape

Enter the no of meanings: 3

Enter the meanings :

A large primate

An unintelligent or clumsy person

an absurd or unthinking way

Do you want to add one more element to the dictionary? If yes, press 1, else press 0: 0

Sample Output:

Here's the dictionary you've created :

Python : ['A large heavy-bodied snake ', 'A high-level programming language']

ape : ['A large primate', 'An unintelligent or clumsy person', 'an absurd or unthinking way']

=====================

There is a method which takes a list of names as input and return a dictionary object.

Each start letter of each name will be the key element of dictionary object, if key already there then increase its occurence value by 1

if not present, just add key with occurence value as 1

return dictionary object

def getDisctionary(listNames):

dict={}

for name in listNames:

if name[0] in dict:

dict[name[0]] = dict[name[0]]+1

else:

dict[name[0]]=1

return dict

dict = getDisctionary(["Amal","Kamal","Azad","Bimal","Karim"])

print(dict)

=============

Problem:

If starting number > ending number (or) starting number<0 (or) ending number<=0, display an Invalid message as specified in the sample output.

If starting number==0, then the roll number should start with 1.

If the step value <0, then starting number should be greater than the ending number.

If the step value >0, then starting number should be less than the ending number.

If step value == 0, then display " Invalid step value"

If start value == end value, then display "Starting and ending number must not be same.

Sample Input and Output statement 1:

Enter the starting number:-567

Invalid starting number

Sample Input and Output statement 2:

Enter the starting number:10

Enter the ending number:0

Invalid ending number

Sample Input and Output statement 3:

Enter the starting number:20

Enter the ending number:100

Enter the step value:25

[20, 45, 70, 95]

============================Done

Problem : There is a list of numbers. We need to pass this list to a function getPerfect()

The method is used to check whether each number in list is perfect ? if yes, add it into a new list

return this list from this method.

define another method isPerfect() which is used to check a given number is perfect number or not.

Example for a perfect number : 28

Perfect number [1+2+4+7+14 == 28 ]

input : [12,28,23,6,18,19,17]

output : [28,6]

def findPerfect(list):

#write code to a new list which contains all perfect numbers only

================

Input :

How many number?

3

Enter number for position[1]:

12

Enter number for position[2]:

22

Enter number for position[1]:

13

Output:

Data are : 12 13 22

Sum : 45

Max : 22

Min : 12

===========================Pattern checking

make program in this way:

def checkPassword(password):

#return string

# password should contain minimum 1 alphabet 1 digit 1 special character (@#!:) , minimum length 6

# if constraints not followed, return "invalid"

# if valid but length within 6 to 10 , return "valid but weak"

# otherwise return "valid and strong"

===================================

define a function to check a given string in proper format as PAN number or not.

def checkPan(panNo):

# pan should contain pattern as first 5 characters will be uppercase letter , next 4 should be digit and last character should be upper case letter

# if it follows rules, return True otherwise return False

===========================================

define a function to check a given string in proper format as Email or not.

def checkEmail(emailId):

# emailId should be in proper email pattern

# if it follows rules, return True otherwise return False

=========== Assignemnts on Class & Object

problem-1:

Define a class : Player

with instance attributes :

playerName , countryName, age

define its constructors to assign some initial values.

Define a method to show details for that particular player.

Define a main class with main-method to make demo for it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Define a class : MyList

It has an attribute as List which stores some integer values.

Define init method to store some data in the list

define str method to show list in this manner (data1,data2,.....)

overload - operators which behaves like between 2 list, it will return only those in a new list as a MyList object where data present in first object but not in second object

Design test methods according to your understandings

=====================================================

Define a class :Complex

attribues : real , imaginary : both are int type

define init & str method to initialize and display complex data

overload + - \* operators to operate between 2 complex objects

======================== Exception labs

Problem-1:

define a function to check whether a given name is valid or not.

for validation :

a. name should be in uppercase letter , includes space only

b. min length 5 , max length 20

if valid , return message as "Name is valid"

otherwise handling an exception with a message {name} is invalid in format

Problem-2:

There is a list of Employee where each employee has information as

{ name: xy,

age: 23,

location: kolkata

}

Wap to check each employee in list, whether their age is between 18 to 60 or not.

At end , return msg as :

{ status : OK

no. of valid employee: x

no. of invalid employee: y

}

if all are invalid , handle it as exception (custom) and return message as

{ status : INVALID

message : All employee are invalid

}

====================================================

WAP to take a positive integer from user and check is it prime or not ?

if not positive, raise an exception with msg "data should be positive"

if other than integer given, raise exception with msg "data is improper format"

===================

WAP to get all prime numbers within a given set of integer.

Check if there is no prime nos, raise an exception with msg "Prime number not found"

Otherwise show all prime numers from the newly created list:

def getPrimes(listNumbers) :

#fill code here

===========================

Given String : AXYZP

Required ouput: BYZAQ

convert : A->B B->C....Y->Z Z->A

Note: only uppercase letter there, if any other , raise exception with msg "improper data"

===================================

define a class Person with 3 attributes : name , age , address

define its constructors , string method (str) to show each property

overload opeator > as follows;

1. first check age

2. if age is same then according to address (in ascending order)

Design test methods according to your understandings

============================================

You are provided with a class Contact with the following attributes as private.

String firstName

String lastName

long phoneNumber

String emailId

A 4 argument constructor and appropriate setters and getters to store and retrieve the details are also provided.

Create a class PhoneBook with attribute

List<Contact> phoneBook = new ArrayList<Contact>();

Write the getters and setters.

Write the following methods in the PhoneBook class.

public void addContact(Contact contactObj) - This method should add the contact object to the phoneBook list.

public List<Contact> viewAllContacts() - This method should return the list of all contacts available.

public Contact viewContactGivenPhone(long phoneNumber) - This method should return the contact details which has the phoneNumber given as parameter.

public boolean removeContact(long phoneNumber) - This method should remove the contact details which has the phoneNumber given as parameter.

If removed return true. Else if the phone number is not available return false.

Write a class Main with the main method. Create the menu as shown in the Sample Input and Output and invoke the corresponding methods as per the choice provided.

===================================

Velan resort decides to automate their bookings. By automating they can easily enter the customer details, number of people visiting the resort, and based on the customer category they can calculate the booking price. So help the resort to do the automation using a java program

Considerations & Requirements:

For 1 adult Rs. 1000 per day

For 1 child Rs. 650 per day

Take input from the user as a single string separating each field with a colon(:)

<customer name>:<number of adults>:<number of child>:<number of days>

If the number of adults is less than zero, print "Invalid input for number of adults"

If the number of children is less than zero, print "Invalid input for number of children"

If the number of days is less than or equal to zero, print "Invalid input for number of days"

Note:

In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.

Ensure to follow the object-oriented specifications provided in the question description.

Ensure to provide the names for classes, attributes, and methods as specified in the question description.

Adhere to the code template, if provided.

Do not use System.exit(0) to terminate the program

Sample Input/Output 1:

Harish:2:2:2

Harish your booking is confirmed and the total cost is Rs 6600

Sample Input/Output 2:

Guru:1:0:0

Invalid input for number of days

Sample Input/Output 3:

Ezhil:1:0:1

Ezhil your booking is confirmed and the total cost is Rs 1000

=======================================

The Sprinter sports club conducted a zone level relay running race, due to the covid situation they conducted the race in a different manner by making the teams perform individually and calculating the time taken by the team to finish the race. The team that takes less time to complete the race will be considered as the winner. Help the sports club to identify their winners by developing a java application.

Assumptions:

Assume each team has 4 members

Take input from the user as a single string with fields separated by a colon (:) as <team name>:<time taken1>:<time taken2>:<time taken3>:<time taken4> and the time taken by the athlete should be in float

Requirements:

If the team number is less than or equal to one, print "Invalid input" and terminate the program

If the time is less than one, print "Invalid number" and terminate the program

The output should be like "<winning team name> team wins the race in <total time taken> minutes"

If two teams have the same time, then who started the race first will be the winning team

Assume the inputs taken from the user will be in the order of the teams which started the race first, second, and so on.

Precision the output time to 2 decimal places, i.e, display the time in the output print statement up to 2 decimal places. For that use, System.out.printf("%.2f", time);

Note:

In the Sample Input / Output provided, the highlighted text in bold corresponds to the input given by the user, and the rest of the text represents the output.

Ensure to follow the object-oriented specifications provided in the question description.

Ensure to provide the names for classes, attributes, and methods as specified in the question description.

Adhere to the code template, if provided.

Do not use System.exit(0) to terminate the program

Sample Input/Output 1:

Enter the number of teams

4

Enter the details

Green:1.10:1.05:1.00:1.02

Red:1.05:1.04:1.10:1.00

Blue:1.11:1.10:1.10:1.05

Yellow:1.09:1.15:1.08:1.10

Green team wins the race in 4.17 minutes

Sample Input/Output 2:

Enter the number of teams

1

Invalid input

Sample Input/Output 3:

Enter the number of teams

2

Enter the details

White:1.02:1.03:0:1.03

Invalid number

=============================

Problem:

Sample Input/Output 1:

Enter the number of teams

4

Enter the details

Green:1.10:1.05:1.00:1.02

Red:1.05:1.04:1.10:1.00

Blue:1.11:1.10:1.10:1.05

Yellow:1.09:1.15:1.08:1.10

Green team wins the race in xxxxxx.xx minutes

Sample Input/Output 2:

Enter the number of teams

1

Invalid input

Sample Input/Output 3:

Enter the number of teams

2

Enter the details

White:1.02:1.03:0:1.03

Invalid number

===============================

Vega organization wants to conduct a cricket tournament. At the end of the tournament, they plan to give a special prize to the player who scored the maximum number of runs. For this, they want an application that can find the maximum runs scored by the players. As a java developer, create a java application to find the maximum score.

Component Specification: ManagementUtility

Type (Class)

Methods

Responsibilities

ManagementUtility

Map<String,Integer> playerMap

Getters and Setters are given in the code skeleton.

ManagementUtility

public void addPlayerscore(String playerName, int score)

This method of the ManagementUtility class to add the playerName as key and score as value to the playerMap

ManagementUtility

public static int maximumScore(Stream<Map.Entry<String,Integer>> playerStream)

This method to find the maximum score from the Stream of map (playerStream).

Note: The class and methods should be declared as public.

Functional requirements:

In the Main class for retrieving the Map<String,Integer> playerMap from the ManagementUtility class, convert the Map<String,Integer> into Stream<Map.Entry<String,Integer>> playerStream. Then pass the playerStream to the maximumScore method in ManagementUtility class and display the maximum score.

In a Main class:

Display the options to choose as "Select an option: 1.Add player score 2.Display 3.Exit" for all iterations. Assume the valid option can be chosen as 1 or 2 or 3.

For option 1: Get the playerName and score as inputs to process the functional requirements and continue to display the options.

For option 2: Retrieve the map as mentioned in the functional requirements. If the retrieved map is empty, then display "No players found" and continue to display the options. Otherwise, display the maximum score as "The maximum score of an individual player for these match is <maximumScore>" and continue to display the options.

For option 3: Display a message "Thank you for using the application" and terminate.

Note:

Do not edit or delete the codes provided in the code template.

Adhere to the Sample Inputs/ Outputs.

In the Sample Inputs/ Outputs provided, the highlighted text in bold corresponds to the input given by the user and the rest of the text represents the output.

Sample Input/ Output 1:

Select an option:

1.Add player score

2.Display

3.Exit

2

No players found

Select an option:

1.Add player score

2.Display

3.Exit

1

Enter the player name

Harper

Enter the score

295

Select an option:

1.Add player score

2.Display

3.Exit

2

The maximum score of an individual player for these match is 295

Select an option:

1.Add player score

2.Display

3.Exit

1

Enter the player name

Mason

Enter the score

320

Select an option:

1.Add player score

2.Display

3.Exit

1

Enter the player name

Avery

Enter the score

310

Select an option:

1.Add player score

2.Display

3.Exit

2

The maximum score of an individual player for these match is 320

Select an option:

1.Add player score

2.Display

3.Exit

3

Thank you for using the application.

Sample Input/ Output 2:

Select an option:

1.Add player score

2.Display

3.Exit

2

No players found

Select an option:

1.Add player score

2.Display

3.Exit

3

Thank you for using the application.

Core Java

=====================================