1. **Object-Oriented programming language:** OOP is a programming paradigm based on concept called “Objects”. It contains data in the form of field called attributes and logic in the form of procedures called methods. An object is an instance created to a class is an abstract blueprint containing data, function logic.

**OOPL** will be totally based on four concepts🡪 Abstraction, Encapsulation, Inheritance, Polymorphism

**Process Oriented programming language**: Procedure-oriented programming basically contains group of instructions known as function. There are multiple functions into the program.

**Difference between process oriented and object-oriented programming language:**

|  |  |
| --- | --- |
| **Process Oriented** | **Object Oriented** |
| Program is divided into small parts called ‘function’ | Program is divided into small parts called ‘objects’ |
| This follows top-down design model where design starts with complete system as a whole and is then divided into small sub-parts. | This follows bottom-up design model where design starts with individual parts and are linked to form a bigger component. |
| There is no access specifier bringing no idea of which class members are accessible to users. | Access specifiers like public, private, protected are available |
| Adding new data and function is not easy because system is already designed | Adding new data and functions are easy |
| Procedural programming is less secure because it doesn’t have any proper way to hide data. | OOP is more secure as it has option of hiding data and access specifiers. |
| Concept of overloading is not possible. | Concept of overloading is possible. |
| POP can’t deal with real world example | OOP deal with real world example |
| Employed in high-level languages like C, FORTRAN, Pascal, Basic | Employed in Java, Python, C++ |

1. **Polymorphism**: If same method is existing multiple times in different forms is called polymorphism.

**Two types of polymorphism**:

1. Static polymorphism or compile time polymorphism: It is achieved by method overloading concept. If same method name exists multiple times with different signature inside the same class, it is called overloading.
2. Dynamic polymorphism or run time polymorphism: It is achieved by method overriding concept. If same method with same prototype is existing in both parent and child class, it is called method overriding.

When calling a method, following two operations are performed:

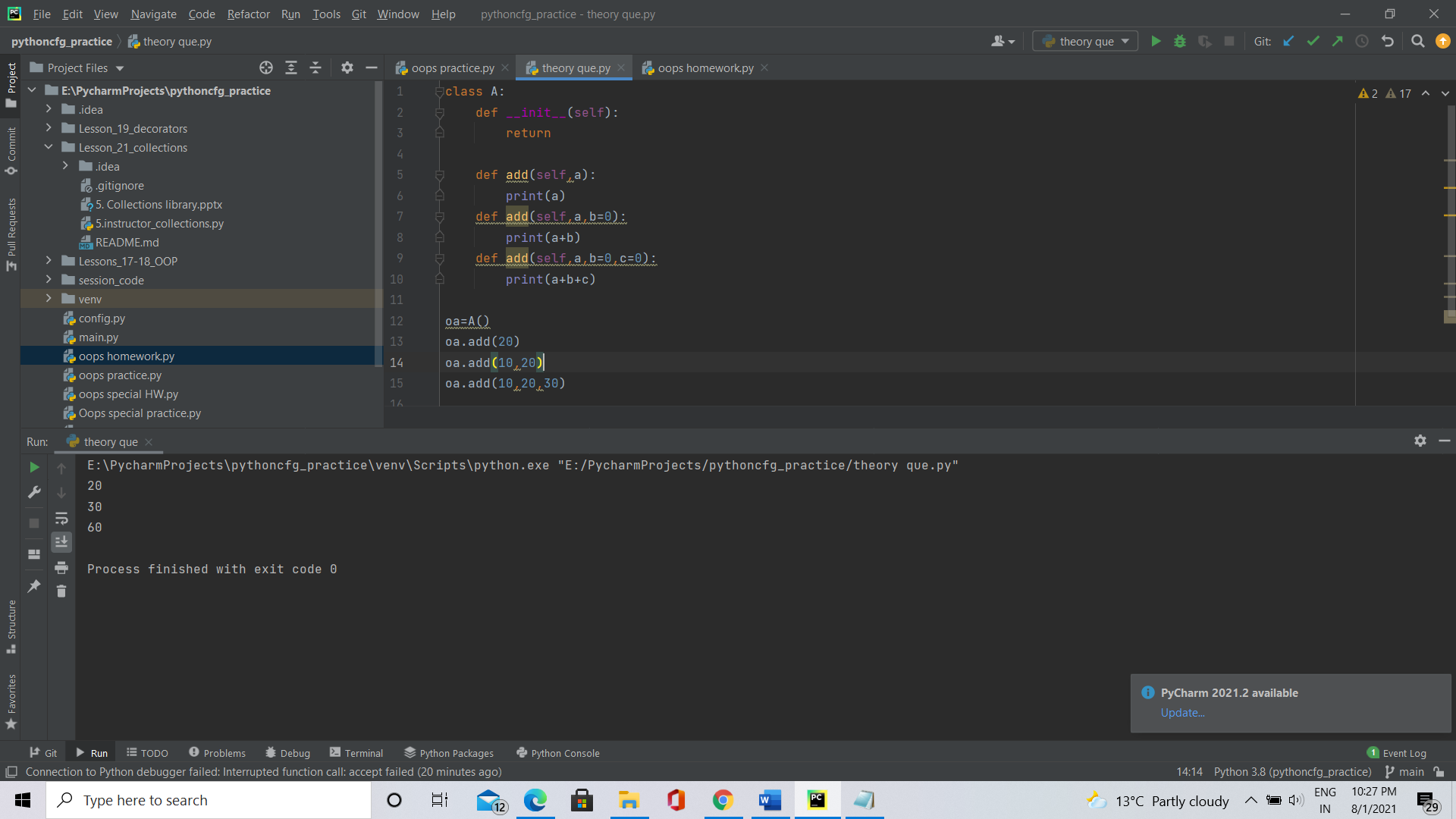
Verifying method at compile time depends on object reference🡺for which class object reference is created, its corresponding method is verified

Executing method at run time depends on object🡺 for which class object is created, its corresponding method is executed.

**Static polymorphism**: If signature is changing 🡺 1. If count or order or type of method parameters are getting changed depending on user’s input, we need to execute method depending on parameters.

Ex:

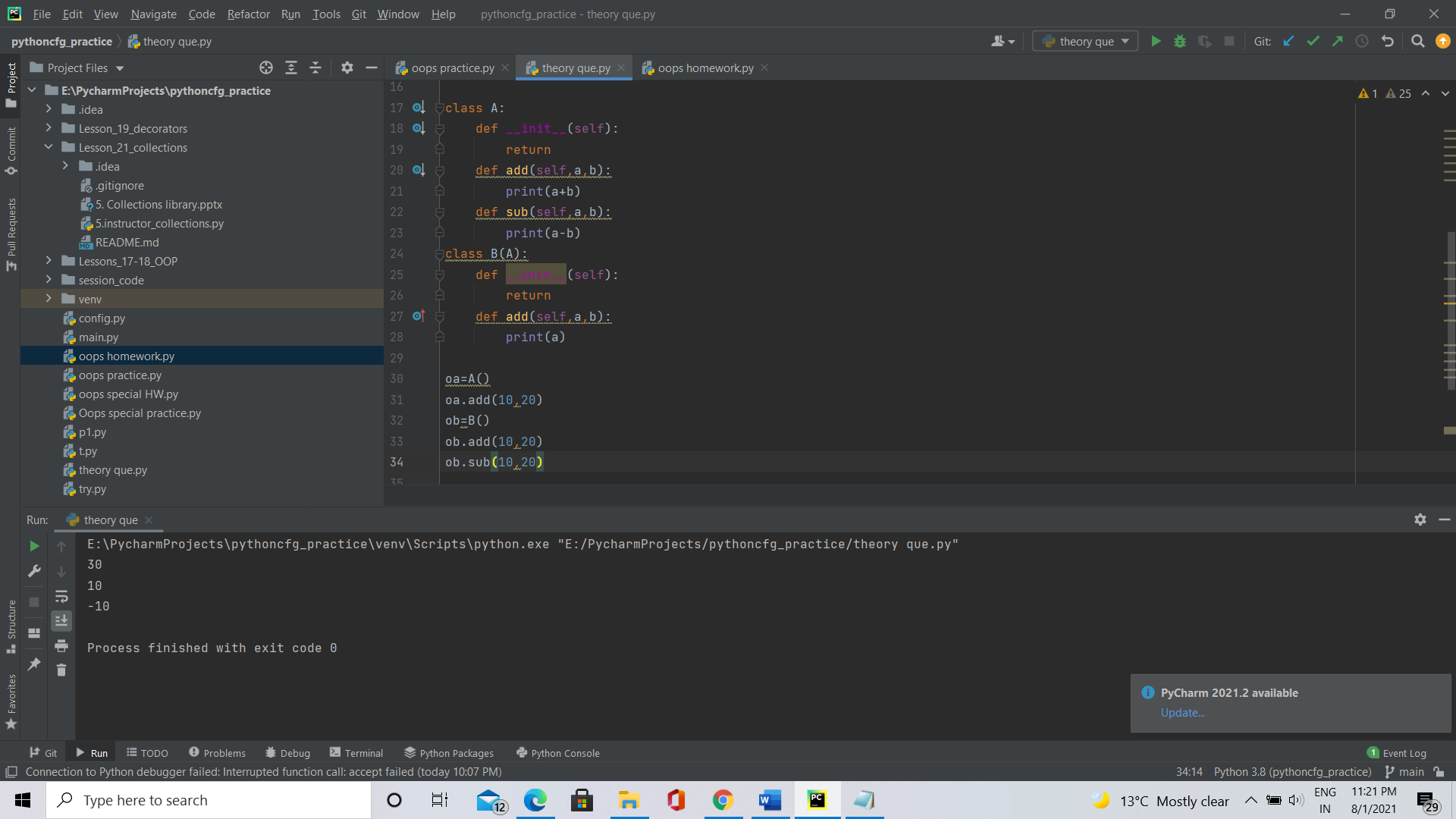
class A:  
 def \_\_init\_\_(self):  
 return  
  
 def add(self,a):  
 print(a)  
 def add(self,a,b=0):  
 print(a+b)  
 def add(self,a,b=0,c=0):  
 print(a+b+c)  
  
oa=A()  
oa.add(20)  
oa.add(10,20)  
oa.add(10,20,30)



Example scenario: While filling a form, users may skip some of the non-mandatory fields, but the developer need to process the form by creating single object to every user whatever may be the fields, method overloading comes into picture.

**Dynamic Polymorphism**: When parent class has multiple children and both parent while child classes have the same method and we need to override the parent method in child class.

class A:  
 def \_\_init\_\_(self):  
 return  
 def add(self,a,b):  
 print(a+b)  
 def sub(self,a,b):  
 print(a-b)  
class B(A):  
 def \_\_init\_\_(self):  
 return  
 def add(self,a,b):  
 print(a)  
  
oa=A()  
oa.add(10,20)  
ob=B()  
ob.add(10,20)  
ob.sub(10,20)



Scenario: When the parent has multiple child classes and the developer don’t know to which child class object is created at run time, dynamic polymorphism comes into picture.

1. **Inheritance:** Inheritance is a concept in object-oriented programming language where one class inherits the attributes and methods of other class. The whose properties got inherited is called parent or super class and the which inherits properties of other class is called child or sub class.

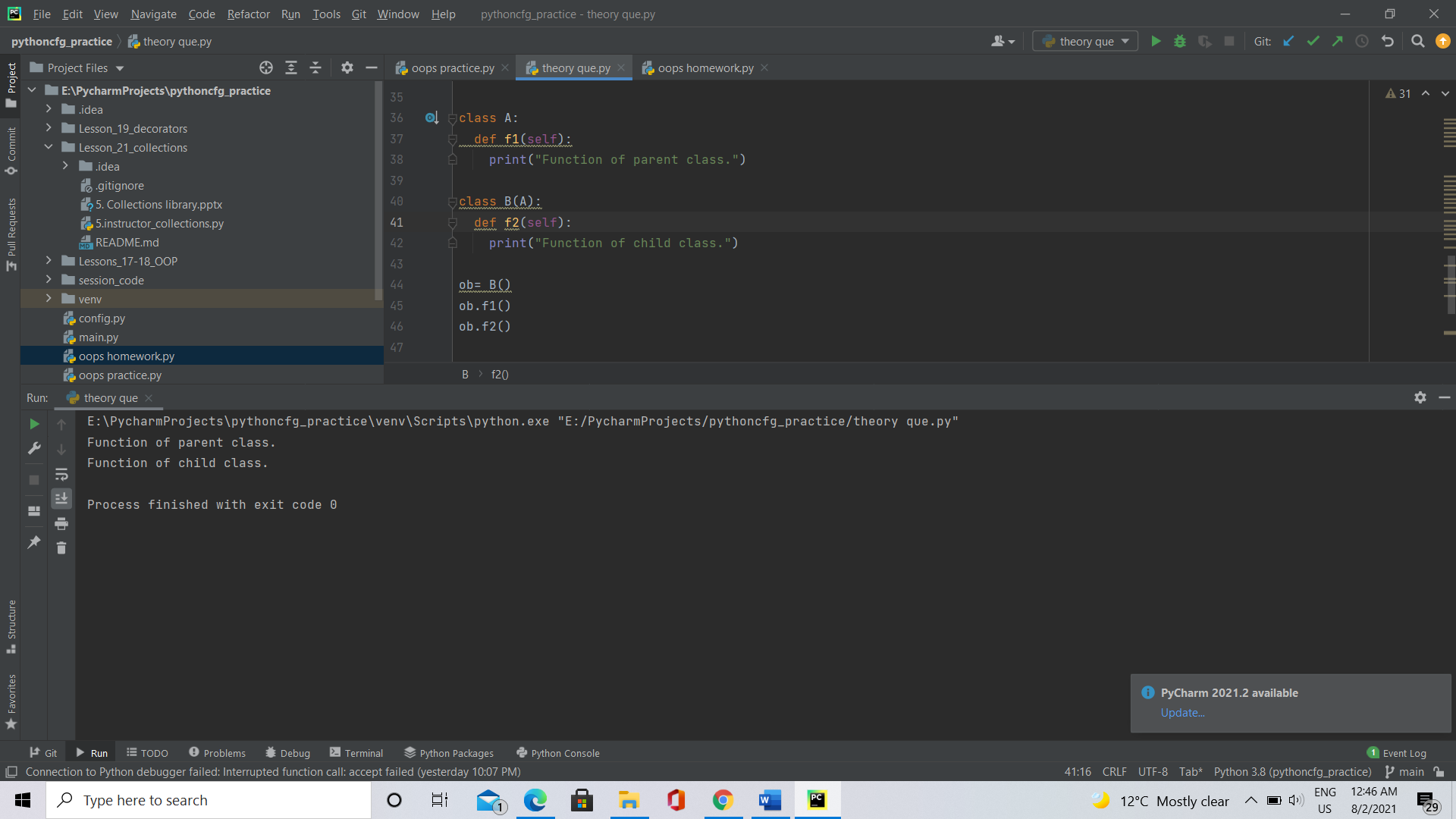
**Note:** Child classes can inherit parent class properties while parent classes cannot inherit child properties. So, parent properties can be called with both parent and child objects references, while child properties can only be called with child object reference.

Inheritance is of five types:

1. Single
2. Multiple
3. Multi-level
4. Hierarchical
5. Hybrid

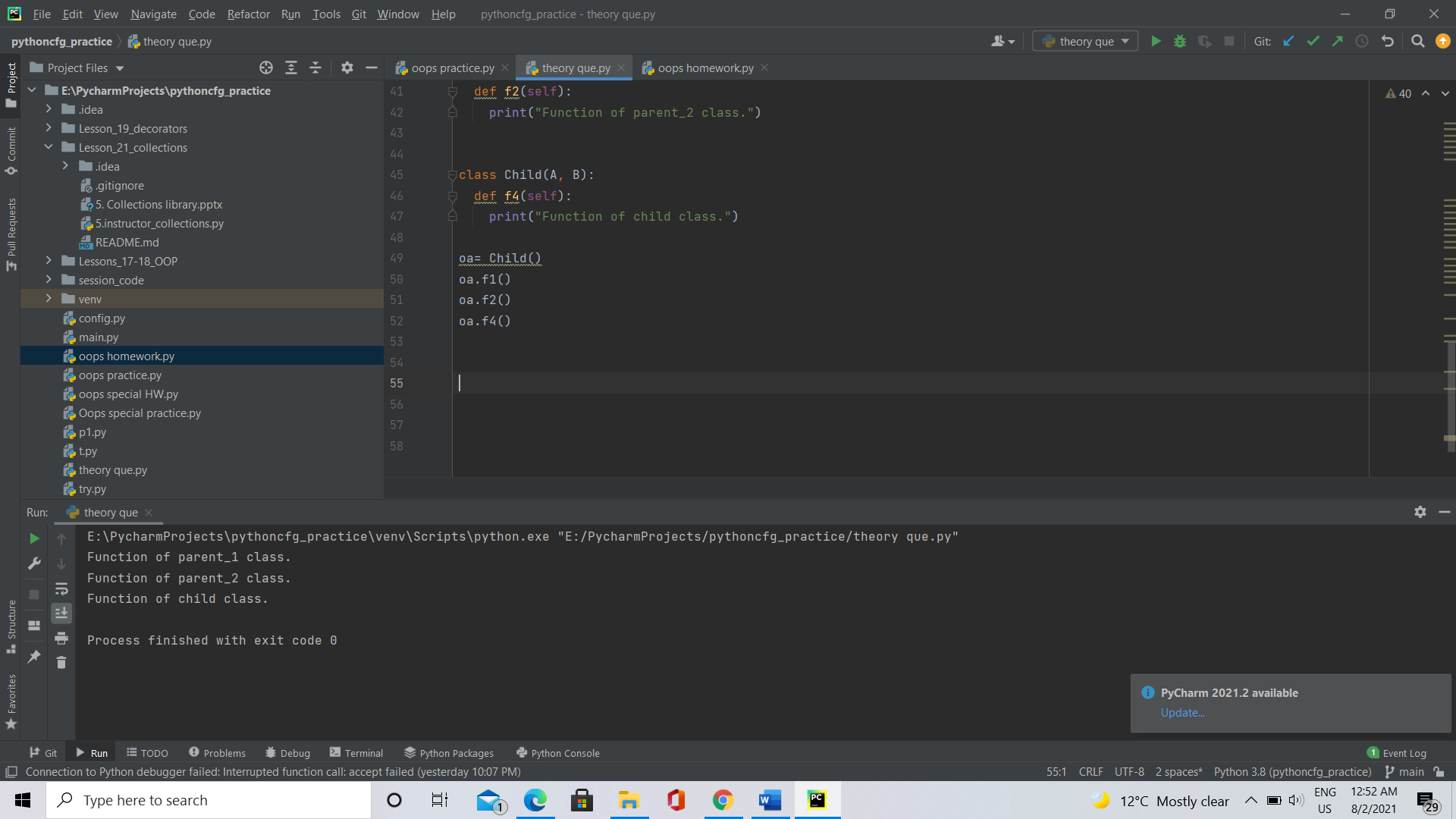
**Single Inheritance:** This is a form of inheritance in which a class inherits only one parent class. This is the simple form of inheritance and hence also referred to as **simple inheritance**.

class A:  
 def f1(self):  
 print("Function of parent class.")  
  
class B(A):  
 def f2(self):  
 print("Function of child class.")  
  
ob= B()  
ob.f1()  
ob.f2()



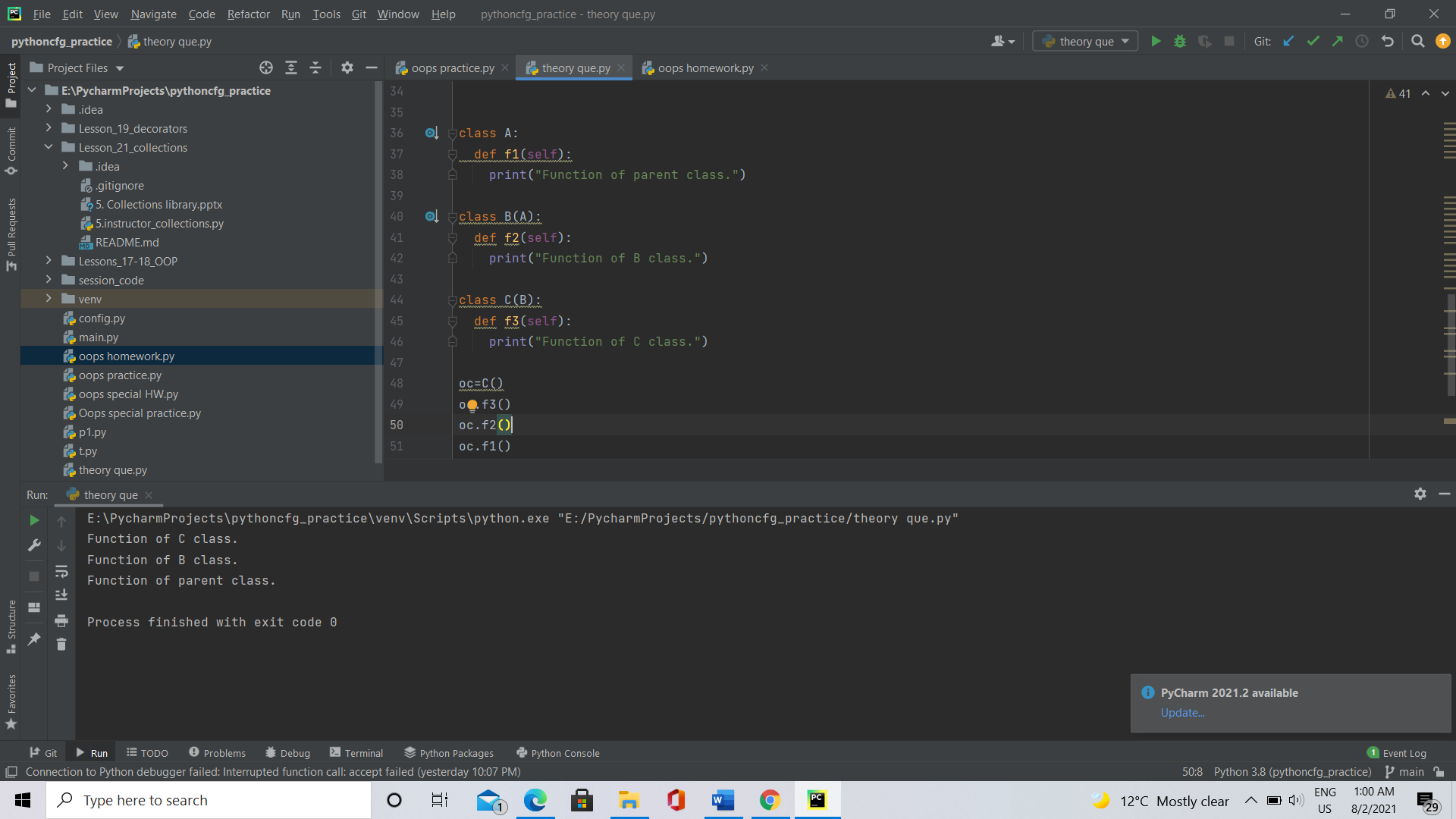
**Multiple inheritance:** In multiple inheritance, single child class inherits from more than one parent class.

class A:  
 def f1(self):  
 print("Function of parent\_1 class.")  
  
class B:  
 def f2(self):  
 print("Function of parent\_2 class.")  
  
  
class Child(A, B):  
 def f4(self):  
 print("Function of child class.")  
  
oa= Child()  
oa.f1()  
oa.f2()  
oa.f4()



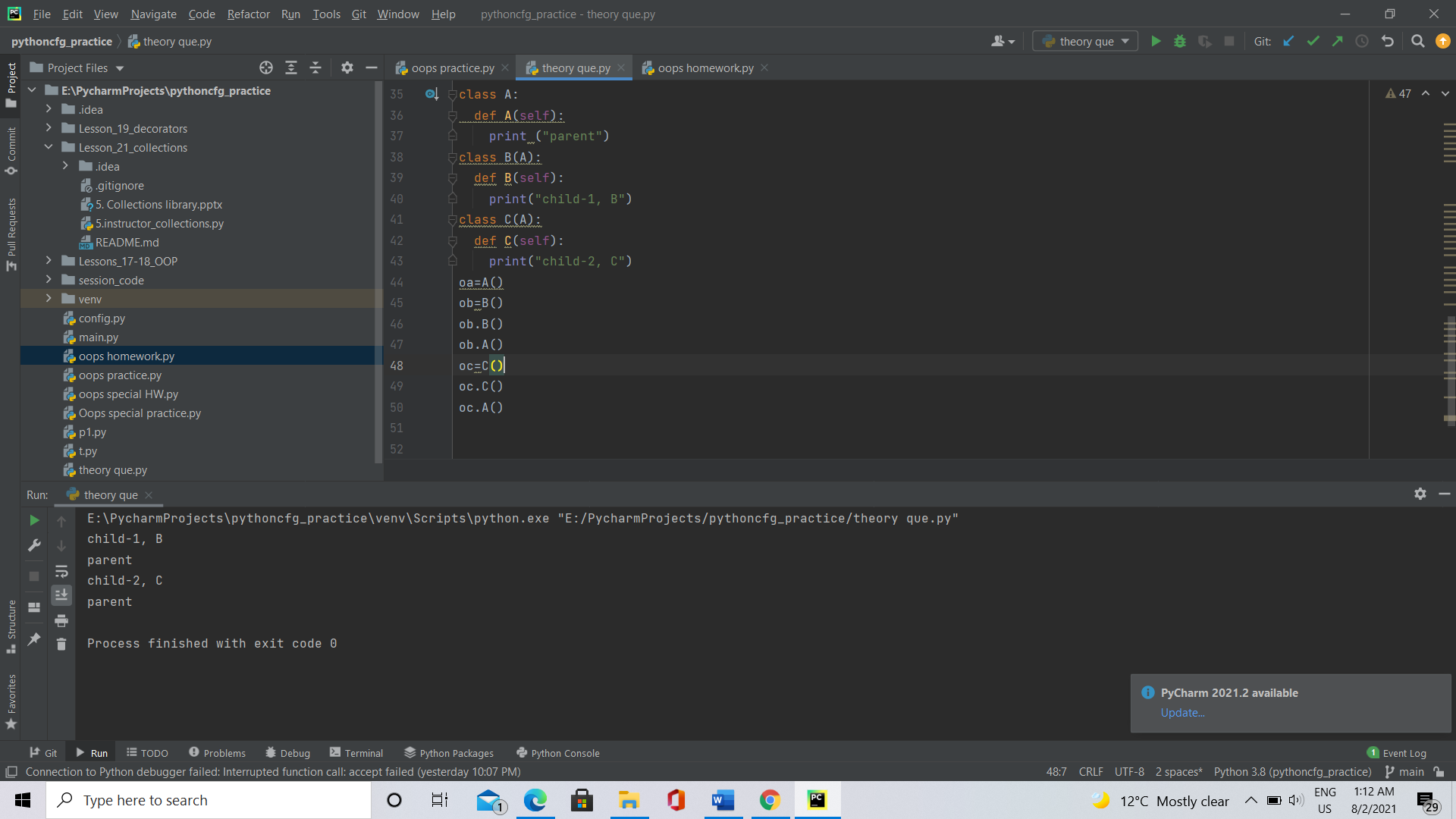
**Multi-level inheritence**: In this inheritence, a child inherets properties from a parent class and this parent class itself been child for another parent class

class A:  
 def f1(self):  
 print("Function of parent class.")  
  
class B(A):  
 def f2(self):  
 print("Function of B class.")  
  
class C(B):  
 def f3(self):  
 print("Function of C class.")  
  
oc=C()  
oc.f3()  
oc.f2()  
oc.f1()



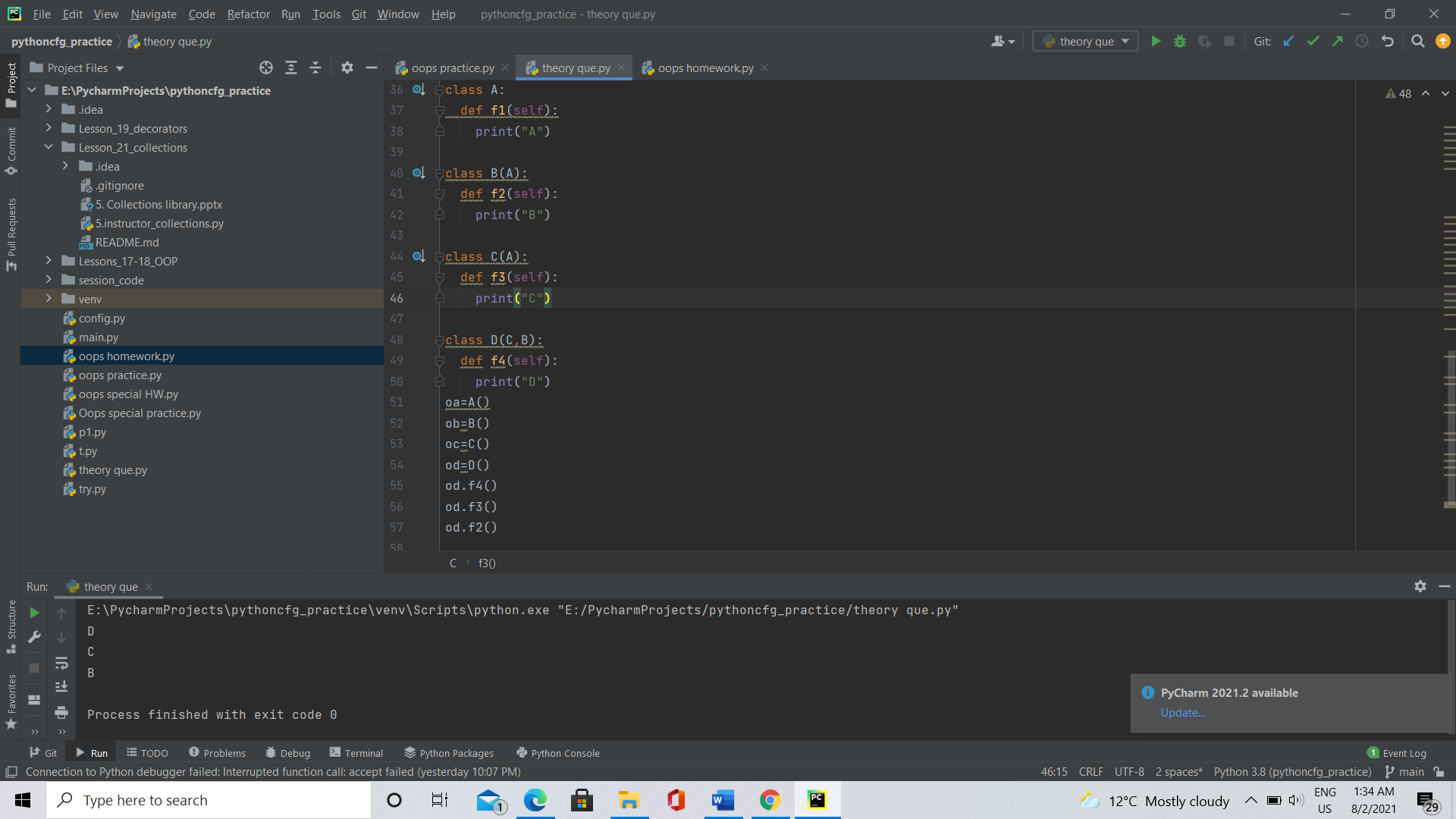
**Hierarchical inheritance**: In this type of inheritance, multiple children inherits from single parent class.

class A:  
 def A(self):  
 print ("parent")  
class B(A):  
 def B(self):  
 print("child-1, B")  
class C(A):  
 def C(self):  
 print("child-2, C")  
oa=A()  
ob=B()  
ob.B()  
ob.A()  
oc=C()  
oc.C()  
oc.A()



**Hybrid Inheritance**: It is combination of more than one form of inheritance.

class A:  
 def f1(self):  
 print("A")  
  
class B(A):  
 def f2(self):  
 print("B")  
  
class C(A):  
 def f3(self):  
 print("C")  
  
class D(C,B):  
 def f4(self):  
 print("D")  
oa=A()  
ob=B()  
oc=C()  
od=D()  
od.f4()  
od.f3()  
od.f2()



1. **Voting for top three funniest people:**

**Procedure:** Every task needs to be started with data collection, implement one-by-one requirement, integrating code snippets finally wrapped with testing.

Intelligence Gathering: First of all, information should be collected regarding funniest people(names, department) in the office.

Analysis: Collected information should be properly analysed and sorted into categories like context of fun, genuinity of information, how many proposed(supporters).

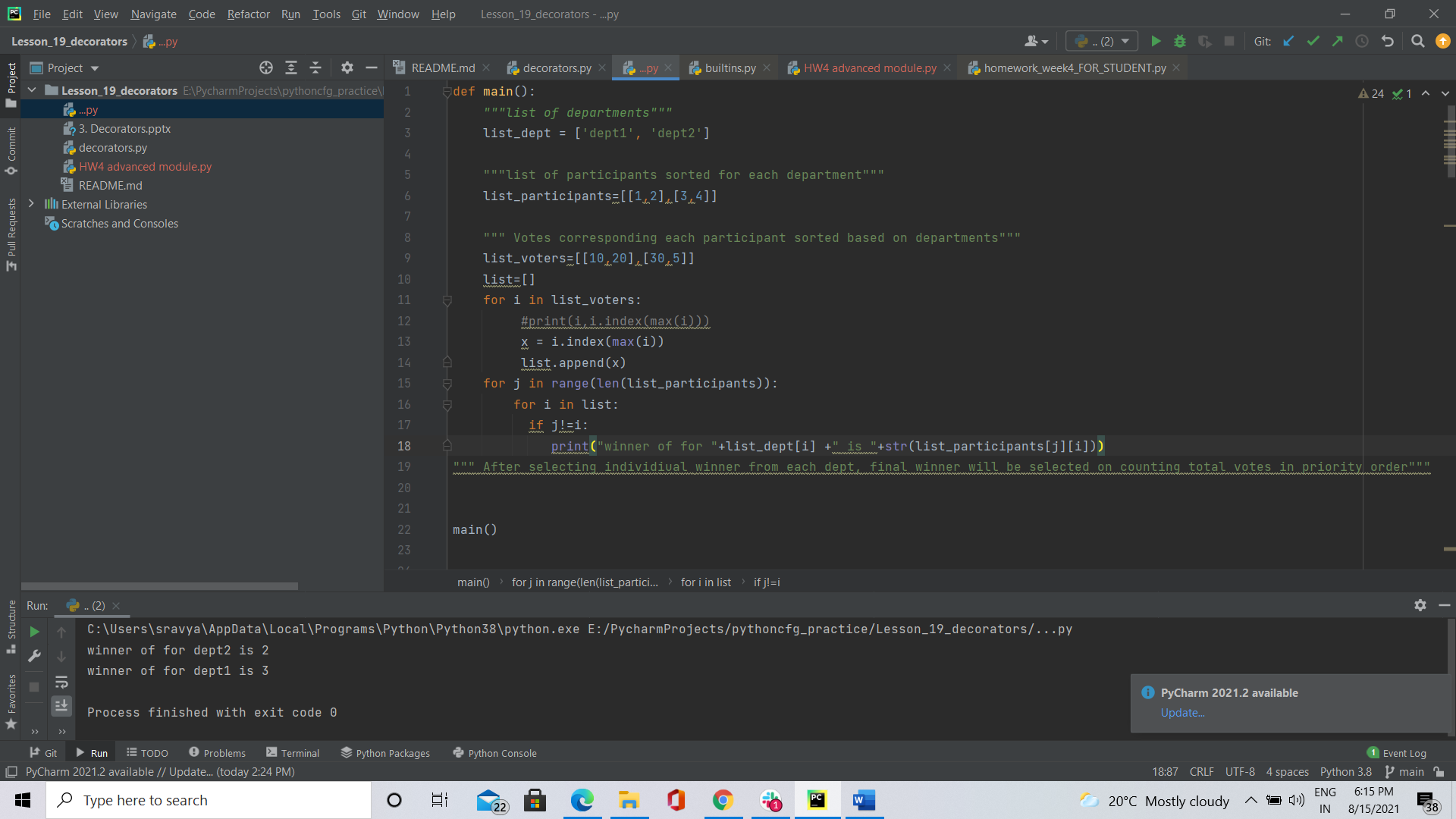
Design and code: Based on information finalized, a program to written aiming to perform voting for funniest people in each department.

If one voter votes more than one people, either he should be asked to vote again or set priority, improper voters should be reminded again to vote.

Finally, the winners from each department can be identified based on votes, reviews and final winner should be selected.

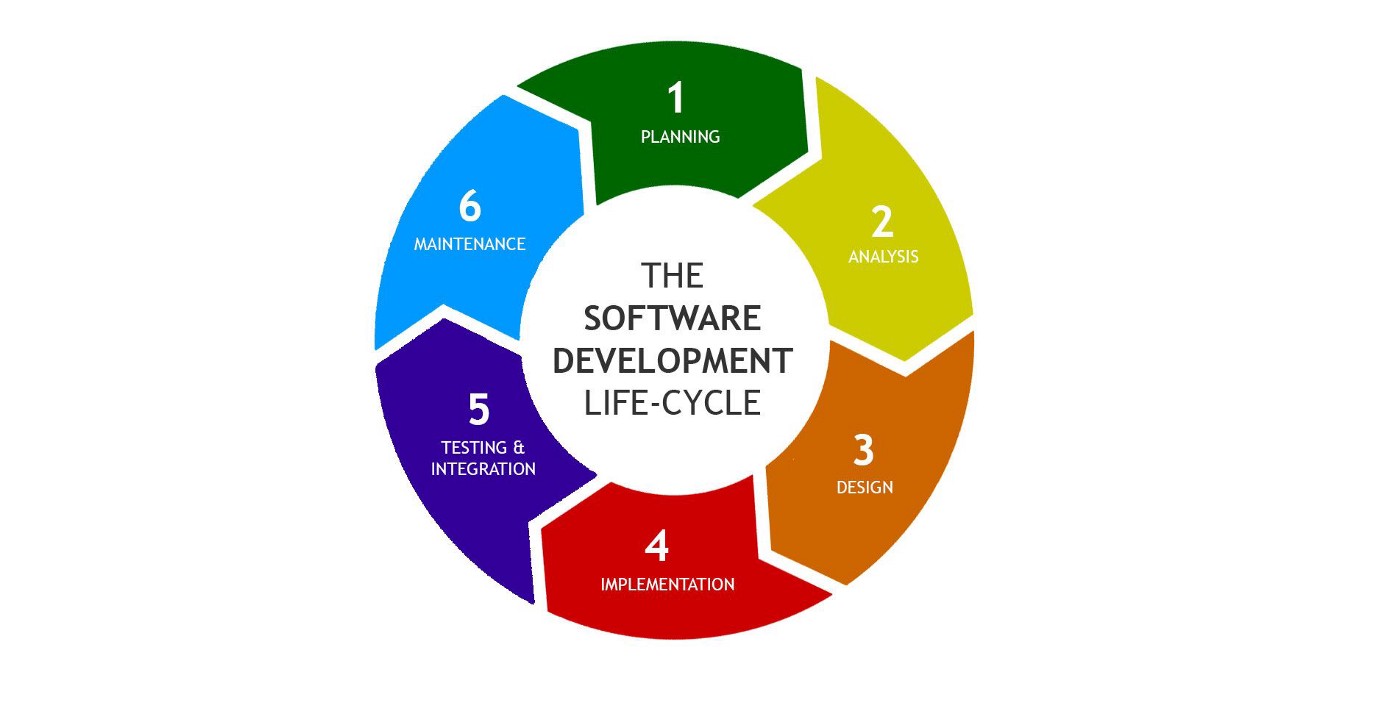
Tested and implemented: Finally integrated program code should be tested before taking to voting.

**Basic code to find funniest person:**



**5.Software Development Life Cycle(SDLC):**

**SDLC** is a systematic process for building software that ensures the quality and correctness of the software built. It is a step-by-step detailed procedure consisting of Planning, defining, designing, building, testing, deployment, maintenance phases and the iteration continues until best featured outcome.



**Planning and collection:**

The requirement is the first stage in the SDLC process. It is conducted by the senior team members by gathering data from product owners and domain experts in the industry. Planning, quality control and recognition of the risks involved is also done at this stage.

This stage gives a clear understanding of the scope of the entire project and the anticipated issues, opportunities associated with the project.

**Analysis:**

The data gathered in planning phase should be analysed and project requirement should be identified and documented.

Requirement include: Budget, Technical needs, Schedule, Legal policies, Personnel

**Design:**

In this phase the system and software design documents are prepared as per the requirement specified.

High-Level Design (HLD)

* Module names and short descriptions and corresponding functionalities
* Interface relationship, link and dependencies between modules
* Database tables identified along with their key elements
* Complete architecture diagrams along with technology details

Low-Level Design (LLD)

* Functional logic of the modules
* Database tables, which include type and size
* Complete detail of the interface
* Addresses all types of dependency issues
* Error messages, input and outputs for every module

**Coding/Implementation:**

Once the system is designed, here comes coding phase. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

**Testing:**

Once the system is completely coded, it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

System is tested by checking each module for functionality, integrated and tested and whole system is again testing to satisfy requirements specified. Some of testing techniques include black box testing (blind testing)—testing is done by random person with no prior knowledge of any functionality, Unit testing---Writing multiple test cases for each functionality, Pair testing—Splitting tasks and combinedly tested by two people, Integration testing---Testing after integrating couple of modules

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

**Deployment (Installation):**

Once the software is made bug/error free, software is about to be deployed. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

**Maintenance:**

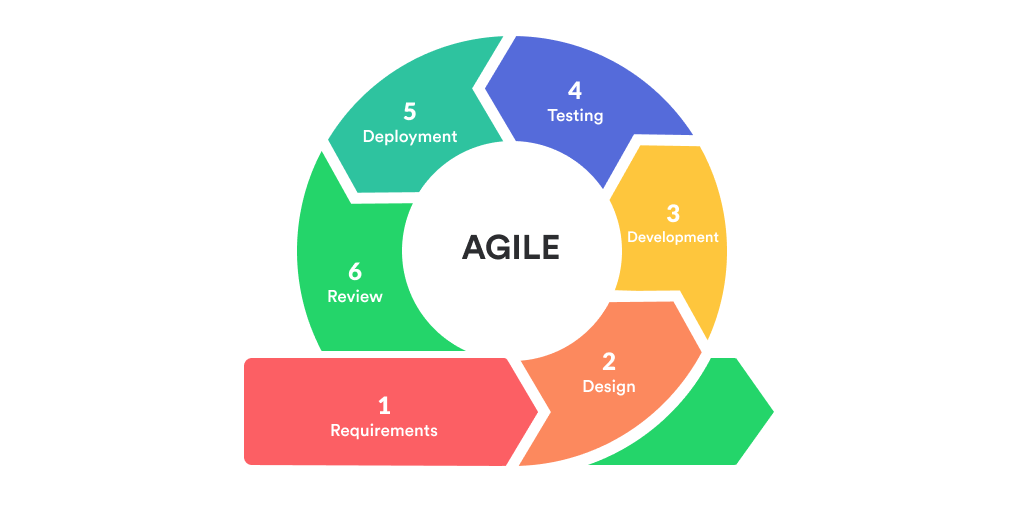
Once the system is deployed, customers get their hands-on system and the developers should continuously maintain system

* Bug fixing - bugs are reported because of some scenarios which are not tested at all
* Updating - Upgrading the application to the newer versions of the Software
* Enhancement - Adding some new features into the existing software

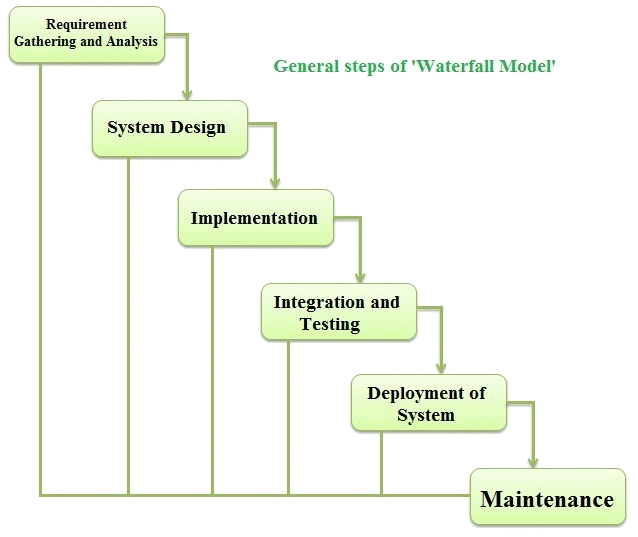
1. Difference between agile and waterfall model

|  |  |
| --- | --- |
| Agile | Waterfall |
| It separates the project development lifecycle into **sprints.** | Software development process is divided into distinct **phases.** |
| Agile follows an incremental approach | Waterfall model follows consecutive design process. |
| It is flexible as requirements can be changed and new ones can be added. | It is structured design, so it can be rigid and is ideal for projects which have definite requirements with no changes |
| It is an iterative based approach because all the phases can be appeared more than once to include changes | It is non-iterative approach, since all the phases are gone through, they are repeated back for more changes. |
| Test plan is reviewed after each sprint | Test plan is rarely discussed in test phase. |
| In this method, testing is performed concurrently with each phase of planning and design | In this model, testing comes only after design phase. |
| It is customer-centric model, project is reviewed and updated every time based on customer reviews | It is project-centric model, cycle continuous until all the project requirements are satisfied. |
| Products owner with team prepares requirements just about every day during a project. | Business analysis prepares requirements before the beginning of the project. |
| Description of project, requirements can be altered anytime during the SDLC process. | Detail description is needed before the implementation waterfall software development approach. |
| Team members and their role are interchangeable based on requirement and no need of project managers, since project is managed by entire team | The process is always straightforward so, project manager plays an essential role |

**Agile Model:**



**Waterfall Model:**



**7.Reduce function:** It is in-built function available in itertools module of python package shown by **reduce()**

**Usage:** reduce() function accepts another function and a sequence as two arguments and returns the result of sequence passed into its argument function.

Accepts only two arguments, one is function and the other is sequence.

**Operation:** At first step, first two elements of sequence are picked and the result is obtained by applying argument function.

Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored until the sequence completes and the final result is printed.

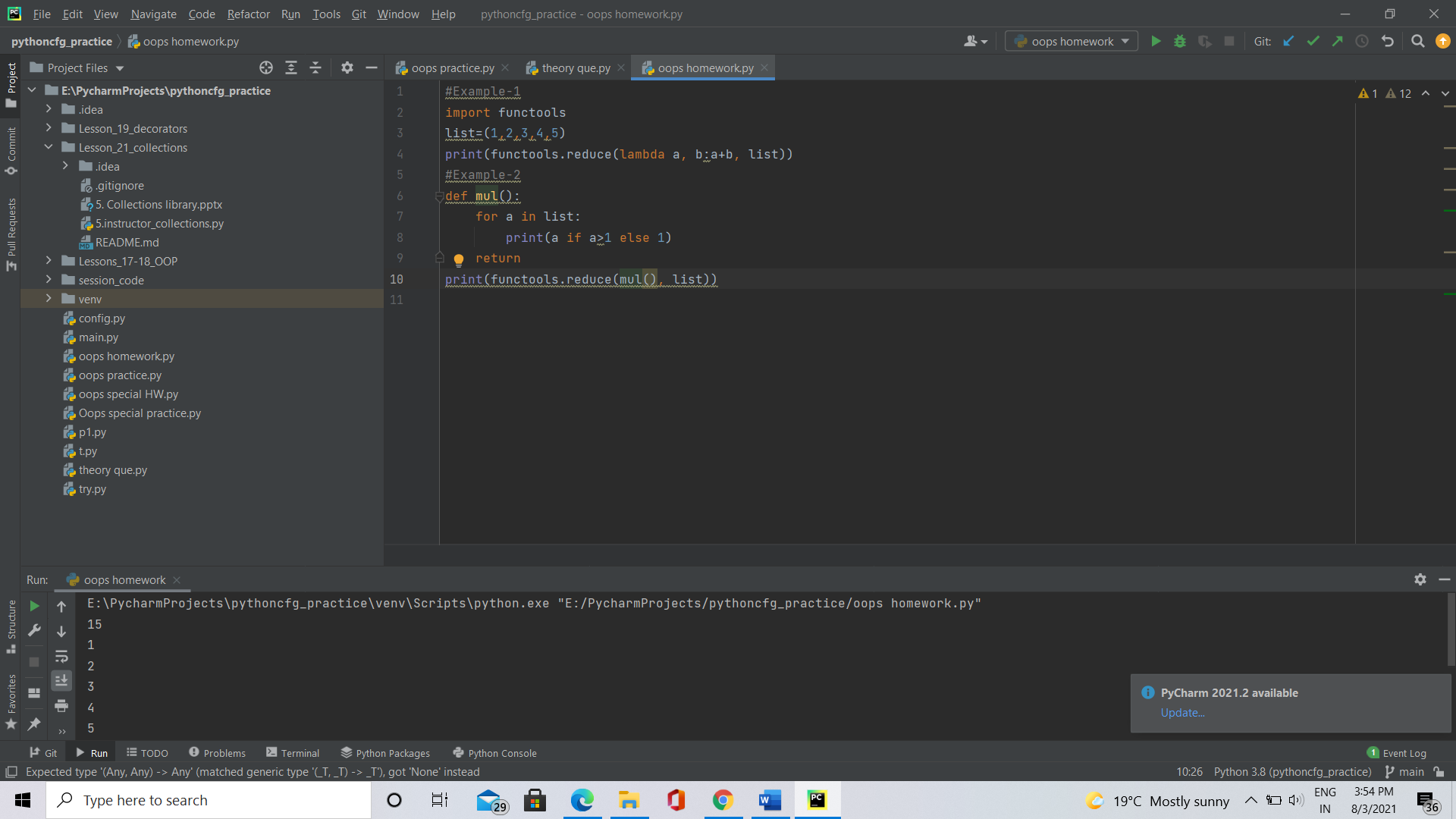
Code:

#Example-1  
import functools  
list=(1,2,3,4,5)  
print(functools.reduce(lambda a, b:a+b, list))  
#Example-2  
def mul():  
 for a in list:  
 print(a if a>1 else 1)  
 return  
print(functools.reduce(mul(), list))

**Explanation: In** the above code snippet, a list [1,2,3,4,5] and a function are passed as two inputs of reduce() function.

Lambda function is a kind of defining a new function, which takes any number of arguments, but only one expression is evaluated and returned.

So, in first example, reduce() function has taken list as arguments and calculated sum of elements of list and returns result 15( show in below picture) and in second example, it again took list as argument, compared each element with a value ‘1’ and returned the result( show below)



**8.Merge Sort:** Merge sort is a divide and conquer algorithm. Merge Sort is a recursive technique where the unsorted elements are divided into two halves/parts and the function calls itself for the parted halves in a manner such that the halves keep recursively dividing themselves into two parts until the entire array is sorted.

It recursively calls itself for the halves or sub-lists until it gets all the elements separated and that no further division is possible i.e., every sub-list contains single element.

# Python program for implementation of MergeSort  
  
# Merges two subarrays of arr[].  
# First subarray is arr[l..m]  
# Second subarray is arr[m+1..r]  
def merge(arr, l, m, r):  
 a1 = m - l + 1  
 a2 = r- m  
  
 # create temp arrays  
 L = [0] \* (a1)  
 R = [0] \* (a2)  
  
 # Copy data to temp arrays L[] and R[]  
 for i in range(0 , a1):  
 L[i] = arr[l + i]  
  
 for j in range(0 , a2):  
 R[j] = arr[m + 1 + j]  
  
 # Merge the temp arrays back into arr[l..r]  
 i = 0 # Initial index of first subarray  
 j = 0 # Initial index of second subarray  
 k = l # Initial index of merged subarray  
  
 while i < a1 and j < a2 :  
 if L[i] <= R[j]:  
 arr[k] = L[i]  
 i += 1  
 else:  
 arr[k] = R[j]  
 j += 1  
 k += 1  
  
 # Copy the remaining elements of L[], if there  
 # are any  
 while i < a1:  
 arr[k] = L[i]  
 i += 1  
 k += 1  
  
 # Copy the remaining elements of R[], if there  
 # are any  
 while j < a2:  
 arr[k] = R[j]  
 j += 1  
 k += 1  
  
# l is for left index and r is right index of the  
# sub-array of arr to be sorted  
def mergeSort(arr,l,r):  
 if l < r:  
  
 # Same as (l+r)//2, but avoids overflow for  
 # large l and h  
 m = (l+(r-1))//2  
  
 # Sort first and second halves  
 mergeSort(arr, l, m)  
 mergeSort(arr, m+1, r)  
 merge(arr, l, m, r)  
  
  
# Driver code to test above  
arr = [1,2,3,10,7,8,9]  
n = len(arr)  
print ("Given array is")  
for i in range(n):  
 print ("%d" %arr[i]),  
  
mergeSort(arr,0,n-1)  
print ("\n\nSorted array is")  
for i in range(n):  
 print ("%d" %arr[i])

Sorted array is

1

2

3

7

8

9

10

**9. Generators:**

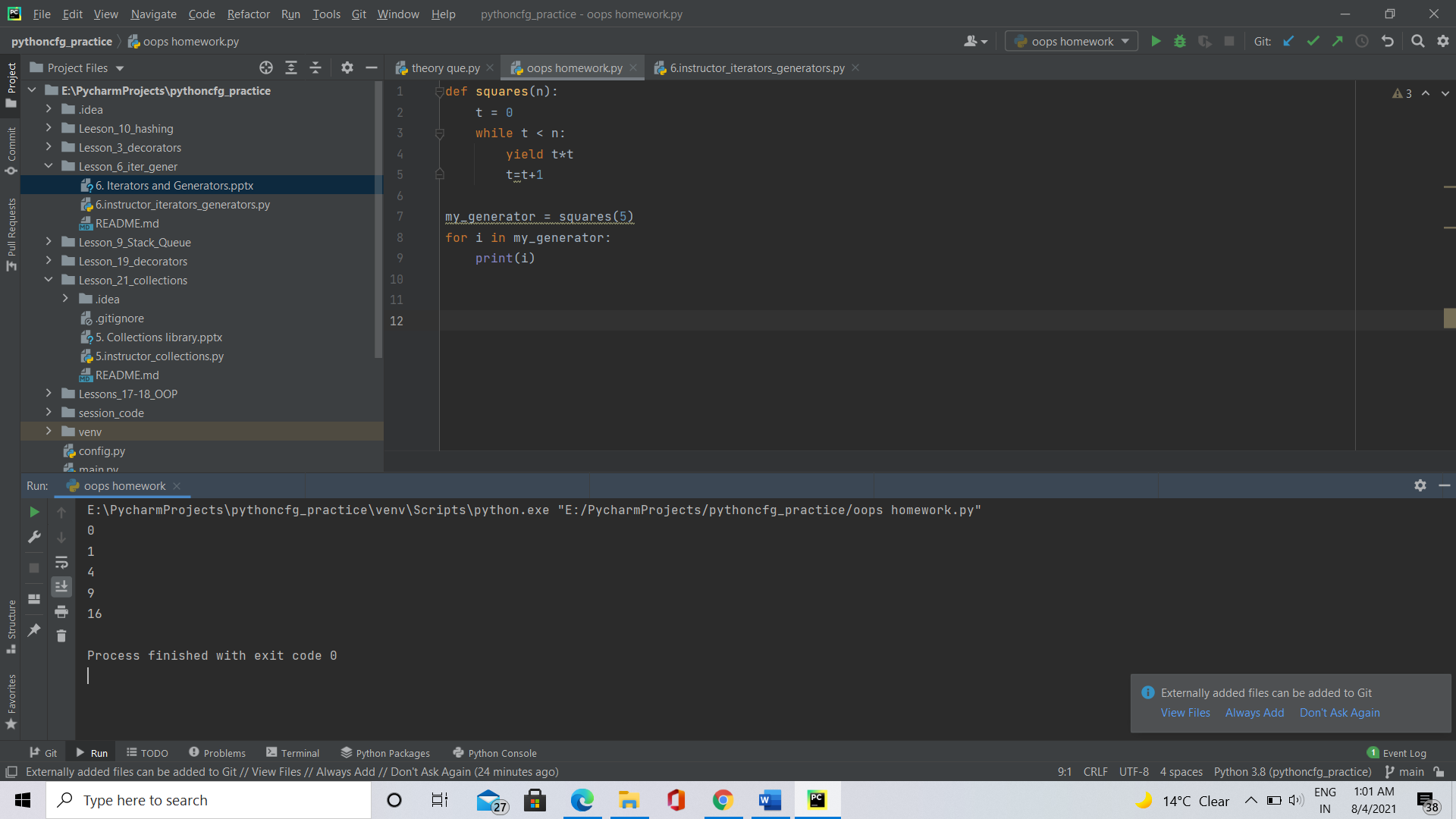
Generators are basically functions that return traversable objects or items. These functions do not produce all the items at once, rather they produce them one at a time and only when required. Generators are used in place of ‘for’ loop to iterate over set of items.

Similar to regular function which use ‘return’ keyword to return a value, generators use ‘yield’ keyword to return value.

Advantages of generators:

* Generators easy to implement as they automatically implement \_\_iter\_\_(), \_\_next\_\_() and Stop Iteration which otherwise, need to be explicitly specified.
* Memory is saved as the items are produced as when required, unlike normal python functions This fact becomes very important when you need to create a huge number of iterators. This is also considered as the biggest advantage of generators.
* Can be used to produce an infinite number of items.
* They can also be used to pipeline a number of operations

**Since, generator lends sequence of values to loop on, without explicitly calling next value or \_\_iter\_\_(), \_\_next()\_\_ functions, generators can be used in a ‘for’ loop.**



Explanation: In the code snippet above, a function is defined to find squares of n numbers and result is returned through yield keyword🡺 generator.

**Since generator has \_\_next\_\_() , iter() inbuilt, there is no need to particular call \_\_next\_\_ function to return next value or iter() for looping**

10. Decorators:

Decorator is a function, to which another function is passed as an argument, enhances the functionality and returns inner argument function.

Decorator generally return inner wrapper function/nested which was used to enhance the functionality of argument function and can be called with ‘sugar’ 🡪 ‘@’

Need for decorators:

1.When the behaviour of function needed to be updated without changing decorator as a whole. Ex: If we are intended to apply permissions, check logs.

2. Decorator code can be written to a file and imported multiple times if needed. This reduces code repetition, redundancy and improves performance.

Basic decorator syntax:

def decorator(): #defining decorating

def inner wrapper(): #defining inner wrapper

function() #calling main function

return inner wrapper

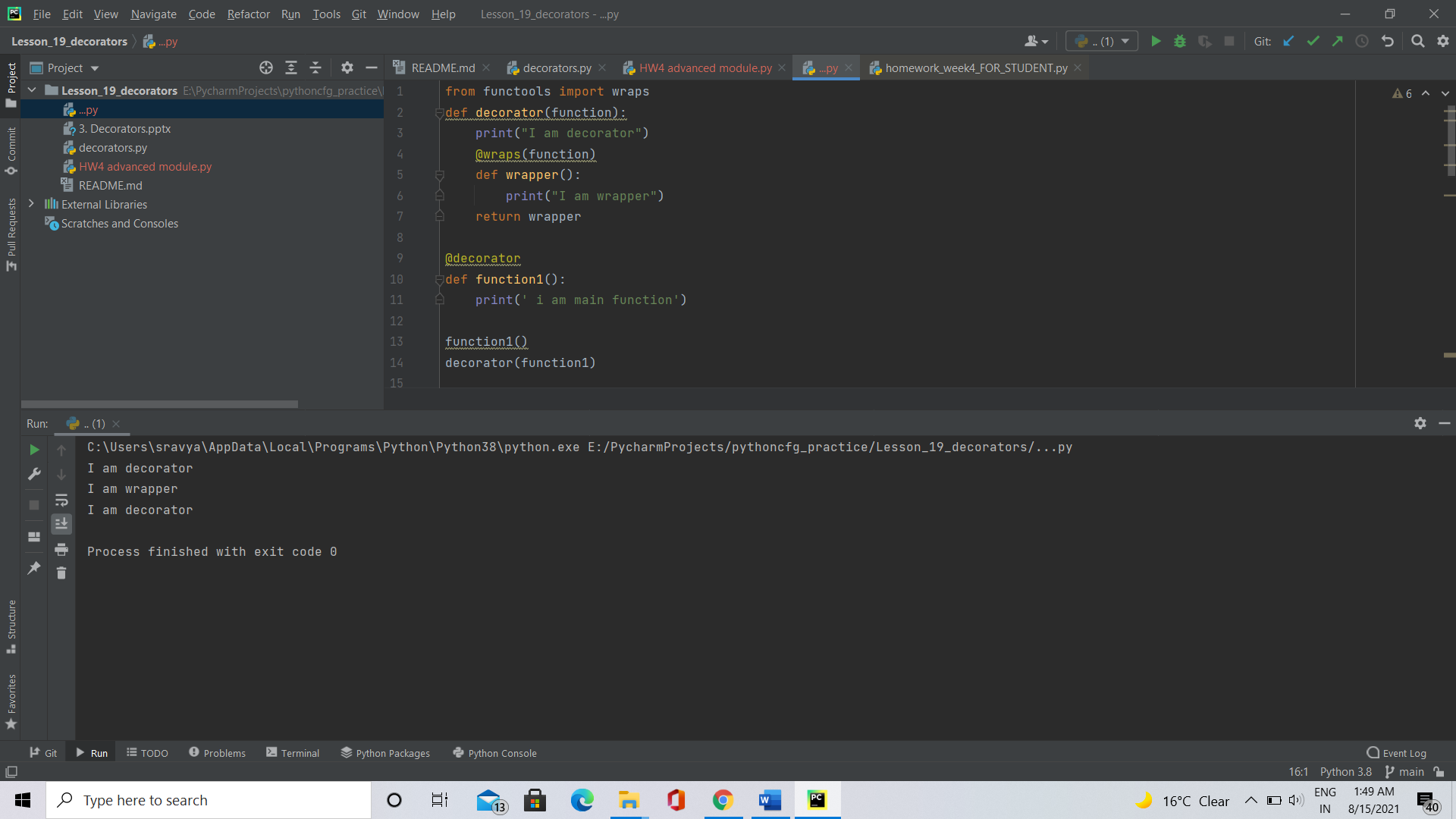
def function(): #define main function

print(“Sravya”)

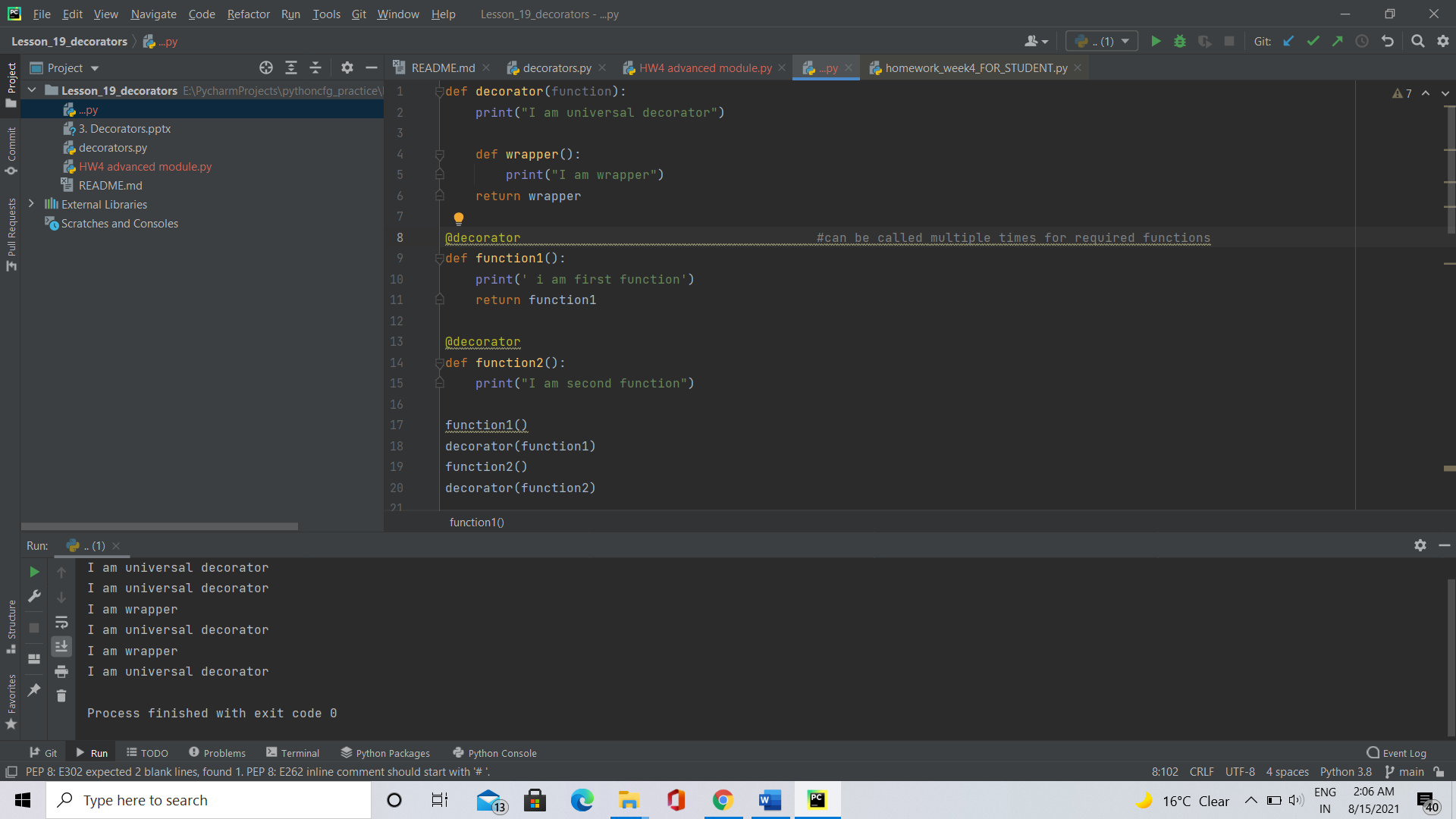
Note: A function can be passed as an argument, returned from another function, called/ defined in another function.

Some widely used decorator types: Simple, Universal, Chained and class as decorators.

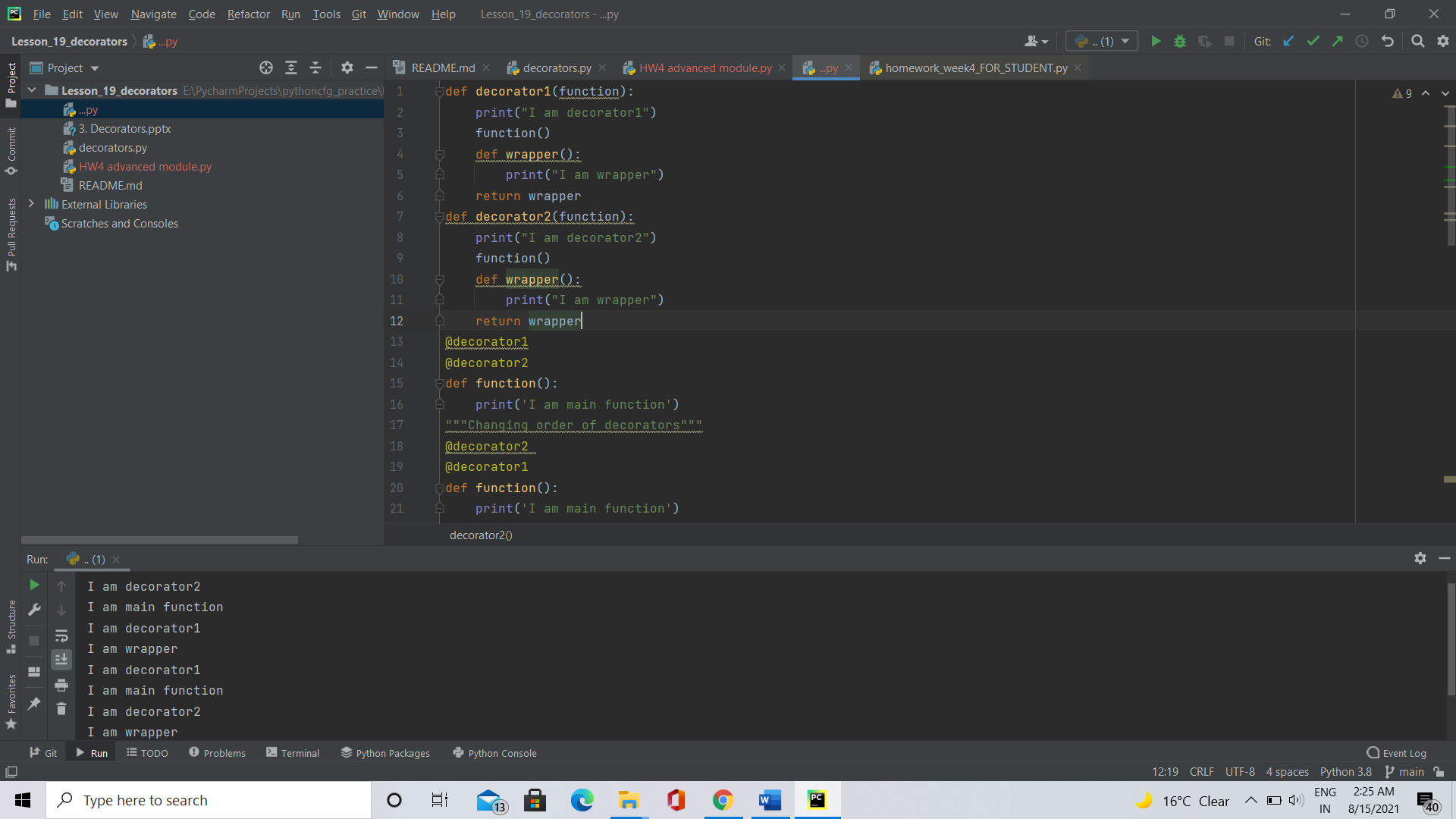
**Simple decorators:** Have a single decorator, wrapper and function to be modified.



**Universal decorators:** Have a decorator, which can be called multiple times, different functions to be improved, one inner wrapper function. For effective use, decorator can be defined in different and imported as a module.

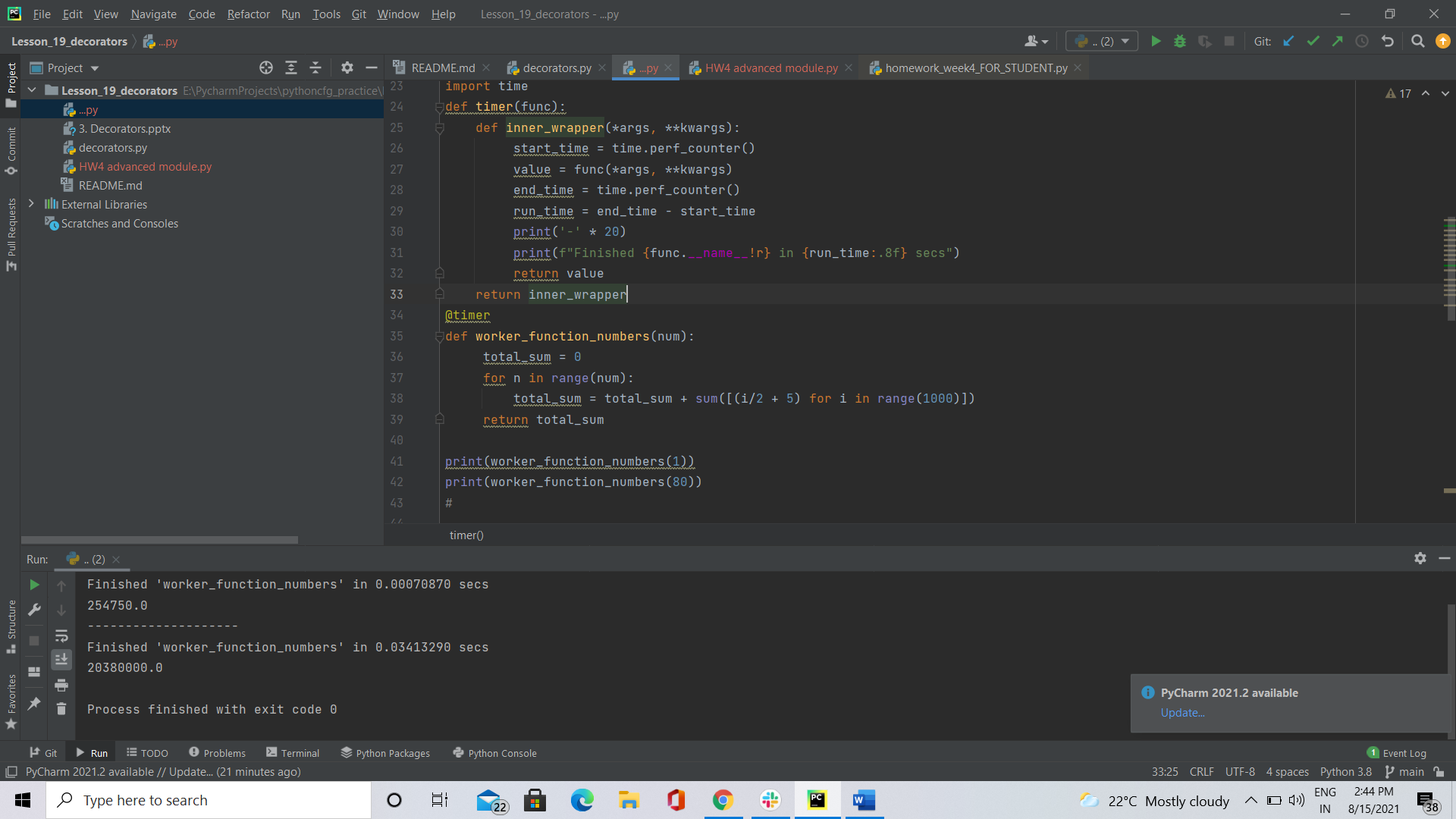


**Chained decorators**: adding multiple features to one function with combination of multiple independent decorators. Simple chained decorator contain multiple decorators chained together, a function and one inner wrapper function.



**Potentially abusive decorators:** Some decorators are not totally vital for a program , but they can an additional feature.

* Timer decorator: It measure the run time required for function to execute decorated function.



* Memory decorator: Stores the states and results in memory class object. When class is created, dynamic object is instantiated and when memory function is called, data stored in its memory can be retrieved.

