|  |  |
| --- | --- |
| Name of Project | Plutus |
| Pronounced as | pluːtəs/ Greek: **Πλοῦτος** |
| History | Plutus was Greek god of wealth |

A picture containing wall

Description automatically generated

**Vision:**

Plutus is an application for MYOB internal payroll team members to view monthly pay slip for a given employee, so that payroll team members don’t have to calculate employee salary manually.

**Scope:**

As a user of plutus,

I should be able to view payroll slip of a given employee in following format:

Monthly Payslip for: “Employee name”

Gross Monthly Income: Gross monthly salary in AUD prefixed with “$”

Monthly Income Tax: Monthly Income tax in AUD prefixed with “$”

Net Monthly Income: Net monthly income in AUD prefixed with “$”

Example:

Monthly Payslip for: “Mary Song”

Gross Monthly Income: $5000

Monthly Income Tax: $500

Net Monthly Income: $4500

So that, I don’t have to calculate employee salary manually.

**Assumptions:**

1. Vishal is PO, BA, Tech lead and Developer for this project
2. There is no need to data ingestion from source
3. In first phase solution will be delivered using static excel spreadsheets having data in them, so tax slab table is constant.
4. In second phase depending on business requirements, solution could be delivered using Kafka/Databases/App integration ..etc
5. Phase 2 is not in scope for this delivery
6. Code is designed to be executed on Mac as of now.
7. Python3.7 is required to execute this code
8. Input CSV contains 1 row per tax slab format. Row format is ‘low\_band\_income\_range’,’high\_band\_income\_range’,’tax\_slab\_rate’.
9. Since sample output does not have decimal, so the decimal part is not required in output.
10. There is no rule for rounding off data.

**Prerequisites:**

1. Docker version 19.03.5 or greater on macbook
2. If behind proxy, Docker proxy configuration is in place. Please see: <https://docs.docker.com/network/proxy/>

**Instructions on how to execute:**

1. Copy or clone the entire project. Cloning can be done from https://github.com/svishal9/plutus
2. Change directory to project root folder which is named as “plutus”
3. Start the application API using the command “./start\_plutus.sh”

|  |
| --- |
| *./start\_plutus.sh* |

1. Execute the command as given in requirement. For example:

|  |
| --- |
| *./GenerateMonthlyPayslip "Mary Song" 60000* |

Sample output:

|  |
| --- |
| Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500 |

1. Once done, then stop the application using the command “./stop\_plutus.sh” as follows:

|  |
| --- |
| *./stop\_plutus.sh* |

**Architecture:**

Infrastructure arch 🡪

A close up of text on a whiteboard

Description automatically generated

Main components of python app 🡪

**A close up of text on a white background

Description automatically generated**

Workflow arch 🡪

**A close up of a sign

Description automatically generated**

**Initial Iteration:**

**Expected behaviour:**

As a business user of plutus,

I should be able to use plutus from console, provide input as employee name and monthly salary, and get payroll details in following format:

Monthly Payslip for: “Mary Song”

Gross Monthly Income: $5000

Monthly Income Tax: $500

Net Monthly Income: $4500

**Additional Information (including Business Logic):**

The annual tax rates are as follows 🡪

|  |  |
| --- | --- |
| Taxable Income | Tax on this income |
| $0 - $20,000 | $0 |
| $20,001 - $40,000 | 10c for each $1 over $20,000 |
| $40,001 - $80,000 | 20c for each $1 over $40,000 |
| $80,0001 - $180,000 | 30c for each $1 over $80,000 |
| $180,001 and over | 40c for each $1 over $180,000 |

For an employee with annual salary of $60,000:

1. Gross monthly income :-- Total annual salary/12 🡺 $60,000/12 = $5000
2. Monthly income tax :-- Calculated as per annual tax rate given above 🡺 ((20,000 \* 0) + ((40,000 – 20,000) \* 0.1) + ((60,000 – 40,000) \* 0.2 ))/12 = (0 + (20,000 \* 0.1) + (20,000 \* 0.2))/12 = (0 + 2,000 + 4,000)/12 = $500
3. Net monthly income :-- Gross monthly income – Monthly income tax 🡺 5,000 – 500 = 4,500

Example console input:

GenerateMonthlyPayslip "Mary Song" 60000

Example Output:

Monthly Payslip for: "Mary Song"

Gross Monthly Income: $5000

Monthly Income Tax: $500

Net Monthly Income: $4500

**Domain Bounded Context Design:**

**A close up of a sign

Description automatically generated**

**High Level Workflow:**

**A sign on a whiteboard

Description automatically generated**

**Standards:**

1. All standards of pylint
2. All variables are in camel case.
3. All Methods start with verb
4. All files and folder names are in lower case separated by “\_”
5. Strictly directed acyclic

**TDD1:**

Create a function to get the tax amount for given annual salary. Test Case:

|  |
| --- |
| **import** GenerateMonthlyPayslip.app **as** GenerateMonthlyPayslip **import** pytest  **def** test\_calculateTax():  **assert** GenerateMonthlyPayslip.GetTax(60000) == 6000, **'Tax calculation not correct'** |

Iteration 1 code:

|  |
| --- |
| taxSlabRanges = [  [0, 20000, 0],  [20000, 40000, 10],  [40000, 80000, 20],  [80000, 180000, 30] ] maxTaxSlabIncome = 180000; maxTaxSlabRate = 40  **def** GetTax( annualIncome ):   annualTaxPayable = []  **for** taxSlab **in** taxSlabRanges:  **if** all([annualIncome > taxSlab[0], annualIncome > taxSlab[1]]):  annualTaxPayable.append((taxSlab[1] - taxSlab[0]) \* taxSlab[2] / 100)  **elif** all([annualIncome > taxSlab[0], annualIncome <= taxSlab[1]]):  annualTaxPayable.append((annualIncome - taxSlab[0]) \* taxSlab[2] / 100)  **if** annualIncome > maxTaxSlabIncome:  annualTaxPayable.append((annualIncome - maxTaxSlabIncome) \* maxTaxSlabRate / 100)  **return** int(sum(annualTaxPayable))  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  print(GetTax(60000)) |

Parameterizing input to calculate tax:

|  |
| --- |
| **import** argparse  taxSlabRanges = [  [0, 20000, 0],  [20000, 40000, 10],  [40000, 80000, 20],  [80000, 180000, 30] ] maxTaxSlabIncome = 180000; maxTaxSlabRate = 40  **def** GetTax( annualIncome ):   annualTaxPayable = []  **for** taxSlab **in** taxSlabRanges:  **if** all([annualIncome > taxSlab[0], annualIncome > taxSlab[1]]):  annualTaxPayable.append((taxSlab[1] - taxSlab[0]) \* taxSlab[2] / 100)  **elif** all([annualIncome > taxSlab[0], annualIncome <= taxSlab[1]]):  annualTaxPayable.append((annualIncome - taxSlab[0]) \* taxSlab[2] / 100)  **if** annualIncome > maxTaxSlabIncome:  annualTaxPayable.append((annualIncome - maxTaxSlabIncome) \* maxTaxSlabRate / 100)  **return** int(sum(annualTaxPayable))  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  parser = argparse.ArgumentParser(description=**'Generate Payslip for an employee'**)  parser.add\_argument(**'--gross-annual-salary'**,**'-gas'**, required=**True**, dest=**'grossAnnualSalary'**, help=**'gross annual salary of employee'**)  args = parser.parse\_args()  grossAnnualSalary=int(args.grossAnnualSalary)  print(grossAnnualSalary)  print(GetTax(grossAnnualSalary)) |

Test case result:

|  |
| --- |
| ========================================================================================================== test session starts ==========================================================================================================  platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1  rootdir: C:\Users\svishal\PycharmProjects\plutus  collected 1 item  tests\unit\test\_u\_app.py . [100%]  =========================================================================================================== 1 passed in 0.04s =========================================================================================================== |

**TDD2:**

CSV file is:

|  |  |  |
| --- | --- | --- |
| low\_band\_income\_range | high\_band\_income\_range | tax\_slab\_rate |
| 0 | 20000 | 0 |
| 20000 | 40000 | 10 |
| 40000 | 80000 | 20 |
| 80000 | 180000 | 30 |
| 180000 | -1 | 40 |

Read from file in above given format and create tax slab ranges, test case is:

|  |
| --- |
| **def** test\_readStaticTaxFile():  maxSlabAnnualIncome, maxSlabTaxRate, tax\_slab\_rate\_list = ReadTaxSlabFile.readTaxFile()  **assert** maxSlabAnnualIncome == 180000, **'Maximum Tax Slab annual income not correct'  assert** maxSlabTaxRate == 40, **'Maximum Tax Slab annual income not correct'  assert** tax\_slab\_rate\_list == [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]] , **'Tax Slab rate is not correct'** |

Iteration 1 code is:

|  |
| --- |
| **import** csv **import** os  **def** readTaxFile():  inputStaticTaxSlabFile = **'StaticInput\\tax\_slab.csv'** tax\_slab\_rate\_list=[]  maxSlabAnnualIncome=0  maxSlabTaxRate=0  **with** open(inputStaticTaxSlabFile) **as** csvfile:  csv\_file = csv.reader(csvfile)  header = csv\_file.\_\_next\_\_()  raw\_data = csv\_file  **for** row **in** raw\_data:  **if**(row[1] == **'-1'**):  maxSlabAnnualIncome = int(row[0])  maxSlabTaxRate = int(row[2])  **else**:  tax\_slab\_list=[int(row[0]),int(row[1]),int(row[2])]  tax\_slab\_rate\_list.append(tax\_slab\_list)  **return** maxSlabAnnualIncome,maxSlabTaxRate,tax\_slab\_rate\_list |

Test results are:

Got few file not found exceptions:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts ==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 2 items*  *tests\unit\test\_u\_app.py .F [100%]*  *=============================================================================================================== FAILURES ================================================================================================================*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_readStaticTaxFile \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *def test\_readStaticTaxFile():*  *> maxSlabAnnualIncome, maxSlabTaxRate, tax\_slab\_rate\_list = ReadTaxSlabFile.readTaxFile()*  *tests\unit\test\_u\_app.py:9:*  *\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*  *def readTaxFile():*  *inputStaticTaxSlabFile = 'StaticInput\\tax\_slab.csv'*  *tax\_slab\_rate\_list=[]*  *maxSlabAnnualIncome=0*  *maxSlabTaxRate=0*  *> with open(inputStaticTaxSlabFile) as csvfile:*  *E FileNotFoundError: [Errno 2] No such file or directory: 'StaticInput\\tax\_slab.csv'*  *GenerateMonthlyPayslip\read\_tax\_slab.py:8: FileNotFoundError*  *====================================================================================================== 1 failed, 1 passed in 0.12s ======================================================================================================* |

Modified code to fix the path of file:

|  |
| --- |
| **import** csv **import** os  **def** readTaxFile():  inputStaticTaxSlabFile = os.getcwd() + **'\\GenerateMonthlyPayslip\\StaticInput\\tax\_slab.csv'** tax\_slab\_rate\_list=[]  maxSlabAnnualIncome=0  maxSlabTaxRate=0  **with** open(inputStaticTaxSlabFile) **as** csvfile:  csv\_file = csv.reader(csvfile)  header = csv\_file.\_\_next\_\_()  raw\_data = csv\_file  **for** row **in** raw\_data:  **if**(row[1] == **'-1'**):  maxSlabAnnualIncome = int(row[0])  maxSlabTaxRate = int(row[2])  **else**:  tax\_slab\_list=[int(row[0]),int(row[1]),int(row[2])]  tax\_slab\_rate\_list.append(tax\_slab\_list)  **return** maxSlabAnnualIncome,maxSlabTaxRate,tax\_slab\_rate\_list |

Test results:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts ==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 2 items*  *tests\unit\test\_u\_app.py .. [100%]*  *=========================================================================================================== 2 passed in 0.06s ===========================================================================================================* |

Refactoring code in GetTax function:

|  |
| --- |
| **def** GetTax( annualIncome ):   annualTaxPayable = []  maxTaxSlabIncome,maxTaxSlabRate, taxSlabRanges = ReadTaxSlabFile.readTaxFile()  **for** taxSlab **in** taxSlabRanges:  **if** all([annualIncome > taxSlab[0], annualIncome > taxSlab[1]]):  annualTaxPayable.append((taxSlab[1] - taxSlab[0]) \* taxSlab[2] / 100)  **elif** all([annualIncome > taxSlab[0], annualIncome <= taxSlab[1]]):  annualTaxPayable.append((annualIncome - taxSlab[0]) \* taxSlab[2] / 100)  **if** annualIncome > maxTaxSlabIncome:  annualTaxPayable.append((annualIncome - maxTaxSlabIncome) \* maxTaxSlabRate / 100)  **return** int(sum(annualTaxPayable)) |

Running tests again:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts ==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 2 items*  *tests\unit\test\_u\_app.py .. [100%]*  *=========================================================================================================== 2 passed in 0.06s ===========================================================================================================* |

TDD3:

Calculate gross monthly salary, net monthly tax payable and net monthly salary given annual salary and annual tax payable. So, the test is as follows:

|  |
| --- |
| **def** test\_calculate\_salary\_details():  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary=SalaryDetails(60000,6000)  **assert** grossMonthlySalary == 5000, **'Gross monthly income not calculated correctly'  assert** monthlyTaxPayable == 500, **'Monthly tax payable not calculated correctly'  assert** netMonthlySalary == 4500, **'Net monthly salary not calculated correctly'** |

Initial code for this test is as follows:

|  |
| --- |
| **def** CalculateNetMonthlyIncome(grossMonthlyIncome,netMonthlyTax):  **return** grossMonthlyIncome-netMonthlyTax  **def** CalculateGrossMonthlyIncome(grossAnnualIncome):  **return** grossAnnualIncome/12  **def** CalculateNetMonthlyTax(annualTaxToBePaid):  **return** annualTaxToBePaid/12  **def** SalaryDetails(grossAnnualIncome,annualTaxToBePaid):  grossMonthlySalary=CalculateGrossMonthlyIncome(grossAnnualIncome)  monthlyTaxPayable=CalculateNetMonthlyTax(annualTaxToBePaid)  netMonthlySalary=CalculateNetMonthlyIncome(grossMonthlySalary,monthlyTaxPayable)  **return** grossMonthlySalary,monthlyTaxPayable,netMonthlySalary |

The test results are as follows:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts* *==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 3 items*  *tests\unit\test\_u\_app.py ... [100%]*  *=========================================================================================================== 3 passed in 0.13s ===========================================================================================================* |

Design Decision 1:

The methods GetAnnualTax and SalaryDetails are objective in nature. They could be used by different processes in future like Finance, HR ..etc. For example, this application just prints the payslip but there could be an application from HR which could use these methods to calculate and insert salary details to their database and analyse further or use it for other application. So, it makes sense to have objective approach towards these methods and plan if they could be reused as library methods instead of class methods. Also, primarily this decision is to expand the scope outside classes as well and showcase use of library methods with classes. In real world scenario this decision may or may not be part of actual solution depending on requirements.

Refactored code:

So far refactored code looks like:

|  |
| --- |
| **import** argparse **from** calculate\_annual\_tax **import** GetAnnualTax **from** calculate\_net\_monthly\_income\_details **import** SalaryDetails  **def** GetEmployeeAnnualTax(grossAnnualSalary):  **return** GetAnnualTax(grossAnnualSalary)  **def** GetEmployeeMonthlyIncomeDetails(grossAnnualSalary,annualTaxToBePaid):  **return** SalaryDetails(grossAnnualSalary,annualTaxToBePaid)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  parser = argparse.ArgumentParser(description=**'Generate Payslip for an employee'**)  parser.add\_argument(**'--gross-annual-salary'**,**'-gas'**, required=**True**, dest=**'grossAnnualSalary'**, help=**'gross annual salary of employee'**)  args = parser.parse\_args()  grossAnnualSalary=int(args.grossAnnualSalary)  print(grossAnnualSalary)  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(grossAnnualSalary,annualTaxPayable)  print(grossMonthlySalary)  print(monthlyTaxPayable)  print(netMonthlySalary) |

TDD4:

Print the salary details, the test case is:

|  |
| --- |
| **def** test\_print\_employee\_monthly\_payslip():  employeeSelected = Employee(**'Mary Song'**,5000,500,4500)  employeeSelected.PrintEmployeePayslip()  expectedOutput = **'Monthly Payslip for: "Mary Song"\nGross Monthly Income: $5000\nMonthly Income Tax: $500\nNet Monthly Income: $4500'  assert** employeeSelected.PrintEmployeePayslip() == expectedOutput , **'Payslip output is not matching as expected'** |

The code to satisfy above test case is as follows:

|  |
| --- |
| **class** Employee:  **def** \_\_init\_\_(self, fullName, grossMonthlySalary, netMonthlyTaxPayable, netMonthlySalary):  self.\_\_employeeFullName = fullName  self.\_\_employeeGrossMonthlySalary = grossMonthlySalary  self.\_\_employeeNetMonthlyTaxPayable = netMonthlyTaxPayable  self.\_\_employeeNetMonthlySalary = netMonthlySalary   **def** PrintEmployeePayslip(self):  payslipOutputMessage=**'Monthly Payslip for: "'** + self.\_\_employeeFullName + **'"\n'** payslipOutputMessage += **'Gross Monthly Income: $'** + str(self.\_\_employeeGrossMonthlySalary) + **'\n'** payslipOutputMessage += **'Monthly Income Tax: $'** + str(self.\_\_employeeNetMonthlyTaxPayable) + **'\n'** payslipOutputMessage += **'Net Monthly Income: $'** + str(self.\_\_employeeNetMonthlySalary)  print(payslipOutputMessage)  **return** payslipOutputMessage |

The test results are:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts ==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 4 items*  *tests\unit\test\_u\_app.py .... [100%]*  *=========================================================================================================== 4 passed in 0.13s ===========================================================================================================* |

Refactoring the code, it looks like as follows:

|  |
| --- |
| **import** argparse **from** calculate\_annual\_tax **import** GetAnnualTax **from** calculate\_net\_monthly\_income\_details **import** SalaryDetails **from** print\_employee\_monthly\_payslip **import** Employee  **def** GetEmployeeAnnualTax(grossAnnualSalary):  **return** GetAnnualTax(grossAnnualSalary)  **def** GetEmployeeMonthlyIncomeDetails(grossAnnualSalary,annualTaxToBePaid):  **return** SalaryDetails(grossAnnualSalary,annualTaxToBePaid)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  parser = argparse.ArgumentParser(description=**'Generate Payslip for an employee'**)  parser.add\_argument(**'--employee-full-name'**,**'-efn'**, required=**True**, dest=**'employeeFullName'**, help=**'Full name of employee'**)  parser.add\_argument(**'--gross-annual-salary'**,**'-gas'**, required=**True**, dest=**'grossAnnualSalary'**, help=**'gross annual salary of employee'**)  args = parser.parse\_args()  employeeFullName=args.employeeFullName  grossAnnualSalary=int(args.grossAnnualSalary)  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(grossAnnualSalary,annualTaxPayable)  employeeSelected=Employee(employeeFullName,grossMonthlySalary, monthlyTaxPayable, netMonthlySalary)  employeeSelected.PrintEmployeePayslip() |

The output looks like:

|  |
| --- |
| C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary Song"  Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000.0  Monthly Income Tax: $500.0  Net Monthly Income: $4500.0 |

It could be clearly seen that the amount has decimal bits which was not in sample output and it was one of assumptions that output is required in int only. So, modified(type casted) the methods for calculation.

|  |
| --- |
| **def** CalculateNetMonthlyIncome(grossMonthlyIncome,netMonthlyTax):  **return** int(grossMonthlyIncome-netMonthlyTax)  **def** CalculateGrossMonthlyIncome(grossAnnualIncome):  **return** int(grossAnnualIncome/12)  **def** CalculateNetMonthlyTax(annualTaxToBePaid):  **return** int(annualTaxToBePaid/12) |

And now the output is:

|  |
| --- |
| C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary Song"  Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500 |

Which is as per expectation.

Looking at unit tests again:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *========================================================================================================== test session starts ==========================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 4 items*  *tests\unit\test\_u\_app.py .... [100%]*  *=========================================================================================================== 4 passed in 0.12s ===========================================================================================================* |

TDD5:

Validate the annual salary input to be integer and greater than 0. Test case is:

|  |
| --- |
| **assert** isinstance(int(grossAnnualSalary),int),**'Gross Annual Salary is not int'** **assert** grossAnnualSalaryInt >= 0, **'Gross Annual Salary cannot be less than 0'** |

Since this condition is always true, added assert:

|  |
| --- |
| **def** ValidateConvertAnnualSalary(grossAnnualSalary):  **assert** isinstance(int(grossAnnualSalary),int),**'Gross Annual Salary is not int'** grossAnnualSalaryInt = int(grossAnnualSalary)  **assert** grossAnnualSalaryInt >= 0, **'Gross Annual Salary cannot be less than 0'  return** grossAnnualSalaryInt |

Refactoring code as:

|  |
| --- |
| **import** argparse **import** logging **from** calculate\_annual\_tax **import** GetAnnualTax **from** calculate\_net\_monthly\_income\_details **import** SalaryDetails **from** print\_employee\_monthly\_payslip **import** Employee **import** logging\_config **from** types **import** \*   log = logging.getLogger(**'generate\_monthly\_payslip.app'**) *# log.setLevel('INFO')* **def** GetEmployeeAnnualTax(grossAnnualSalary):  **return** GetAnnualTax(grossAnnualSalary)  **def** GetEmployeeMonthlyIncomeDetails(grossAnnualSalary,annualTaxToBePaid):  **return** SalaryDetails(grossAnnualSalary,annualTaxToBePaid)  **def** ValidateConvertAnnualSalary(grossAnnualSalary):  **assert** isinstance(int(grossAnnualSalary),int),**'Gross Annual Salary is not int'** grossAnnualSalaryInt = int(grossAnnualSalary)  **assert** grossAnnualSalaryInt >= 0, **'Gross Annual Salary cannot be less than 0'  return** grossAnnualSalaryInt  **def** Run(employeeFullName,grossAnnualSalaryInput):  grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(grossAnnualSalary,annualTaxPayable)  employeeSelected=Employee(employeeFullName,grossMonthlySalary, monthlyTaxPayable, netMonthlySalary)  employeeSelected.PrintEmployeePayslip()  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  log.info(**'Welcome to the world of "Plutus". Starting payslip generator application.'**)  parser = argparse.ArgumentParser(description=**'Generate Payslip for an employee'**)  parser.add\_argument(**'--employee-full-name'**,**'-efn'**, required=**True**, dest=**'employeeFullName'**, help=**'Full name of employee'**)  parser.add\_argument(**'--gross-annual-salary'**,**'-gas'**, required=**True**, dest=**'grossAnnualSalary'**, help=**'gross annual salary of employee'**)  log.debug(**'Parsing parameters'**)  args = parser.parse\_args()  Run(args.employeeFullName,args.grossAnnualSalary) |

The test results are as follows:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Monthly Payslip for: "Mary Song"*  *Gross Monthly Income: $5000*  *Monthly Income Tax: $500*  *Net Monthly Income: $4500*  *(untitled) C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=-60000 -efn="Mary Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Traceback (most recent call last):*  *File "GenerateMonthlyPayslip\app.py", line 39, in <module>*  *Run(args.employeeFullName,args.grossAnnualSalary)*  *File "GenerateMonthlyPayslip\app.py", line 26, in Run*  *grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)*  *File "GenerateMonthlyPayslip\app.py", line 22, in ValidateConvertAnnualSalary*  *assert grossAnnualSalaryInt >= 0, 'Gross Annual Salary cannot be less than 0'*  *AssertionError: Gross Annual Salary cannot be less than 0*  *(untitled) C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=-6a0 -efn="Mary Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Traceback (most recent call last):*  *File "GenerateMonthlyPayslip\app.py", line 39, in <module>*  *Run(args.employeeFullName,args.grossAnnualSalary)*  *File "GenerateMonthlyPayslip\app.py", line 26, in Run*  *grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)*  *File "GenerateMonthlyPayslip\app.py", line 20, in ValidateConvertAnnualSalary*  *assert isinstance(int(grossAnnualSalary),int),'Gross Annual Salary is not int'*  *ValueError: invalid literal for int() with base 10: '-6a0'* |

TDD6:

Validate employee name, test case is:

**assert** name.isalpha(),**'Name can contain only alphabets and spaces'**

The code for this test case is:

|  |
| --- |
| **def** ValidateName(fullName):  **for** name **in** fullName.split(**' '**):  print(name)  **assert** name.isalpha(),**'Name can contain only alphabets and spaces'** |

Changed Run() method to refactor the code:

|  |
| --- |
| **def** ValidateName(fullName):  **for** name **in** fullName.split(**' '**):  print(name)  **assert** name.isalpha(),**'Name can contain only alphabets and spaces'    def** Run(employeeFullName,grossAnnualSalaryInput):  ValidateName(employeeFullName)  grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(grossAnnualSalary,annualTaxPayable)  employeeSelected=Employee(employeeFullName,grossMonthlySalary, monthlyTaxPayable, netMonthlySalary)  employeeSelected.PrintEmployeePayslip() |

Test results are:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Mary*  *Song*  *Monthly Payslip for: "Mary Song"*  *Gross Monthly Income: $5000*  *Monthly Income Tax: $500*  *Net Monthly Income: $4500*  *(untitled) C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary-Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Mary-Song*  *Traceback (most recent call last):*  *File "GenerateMonthlyPayslip\app.py", line 47, in <module>*  *Run(args.employeeFullName,args.grossAnnualSalary)*  *File "GenerateMonthlyPayslip\app.py", line 33, in Run*  *ValidateName(employeeFullName)*  *File "GenerateMonthlyPayslip\app.py", line 28, in ValidateName*  *assert name.isalpha(),'Name can contain only alphabets and spaces'*  *AssertionError: Name can contain only alphabets and spaces*  *(untitled) C:\Users\svishal\PycharmProjects\plutus>python GenerateMonthlyPayslip\app.py -gas=60000 -efn="Mary\ Song"*  *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Mary\*  *Traceback (most recent call last):*  *File "GenerateMonthlyPayslip\app.py", line 47, in <module>*  *Run(args.employeeFullName,args.grossAnnualSalary)*  *File "GenerateMonthlyPayslip\app.py", line 33, in Run*  *ValidateName(employeeFullName)*  *File "GenerateMonthlyPayslip\app.py", line 28, in ValidateName*  *assert name.isalpha(),'Name can contain only alphabets and spaces'*  *AssertionError: Name can contain only alphabets and spaces* |

TDD7:

Validate the input tax slab file, following conditions makes sense for now:

1. All data in tax slab excel spreadsheet should be int apart from header.
2. Data read from the tax slab should have only following static information:

|  |  |  |
| --- | --- | --- |
| low\_band\_income\_range | high\_band\_income\_range | tax\_slab\_rate |
| 0 | 20000 | 0 |
| 20000 | 40000 | 10 |
| 40000 | 80000 | 20 |
| 80000 | 180000 | 30 |
| 180000 | -1 | 40 |

1. high\_band\_income\_range for max tax bracket is -1, this is used only as a flag.

test cases are:

|  |
| --- |
| **assert** isinstance(int(eachTaxSlabRow[0]),int) **and** isinstance(int(eachTaxSlabRow[1]), int) **and** isinstance(int(eachTaxSlabRow[2]), int),**'All data in Tax Slab file should be int apart from header row'  assert** eachTaxSlabRow[0] >= 0 **and** eachTaxSlabRow[1] > 0 **and** eachTaxSlabRow[2] >= 0 , **'Lower income bracket and tax rate should be greater than or equal to 0. Upper income bracket should be greater than 0.'**  expectedOutput = [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]]  **assert** entireTaxSlabWithoutMaximumIncome == expectedOutput,**'Tax Slab does not match with expected output.'**  **assert** maxSlabAnnualIncome == 180000 **and** maxSlabTaxRate == 40 , **'Maximum tax slab does not match with expected output'**  **assert** validTaxSlabWithoutMaximumIncome **and** validTaxSlabMaximumIncome , **'Tax slab data is not valid'** |

So, the code which would facilitate the test conditions is:

|  |
| --- |
| **def** ValidateEachTaxSlab(eachTaxSlabRow):  **assert** isinstance(int(eachTaxSlabRow[0]),int) **and** isinstance(int(eachTaxSlabRow[1]), int) **and** isinstance(int(eachTaxSlabRow[2]), int),**'All data in Tax Slab file should be int apart from header row'  assert** eachTaxSlabRow[0] >= 0 **and** eachTaxSlabRow[1] > 0 **and** eachTaxSlabRow[2] >= 0 , **'Lower income bracket and tax rate should be greater than or equal to 0. Upper income bracket should be greater than 0.'  return True  def** ValidateTaxSlabWithoutMaximumIncome(entireTaxSlabWithoutMaximumIncome):  expectedOutput = [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]]  **assert** entireTaxSlabWithoutMaximumIncome == expectedOutput,**'Tax Slab does not match with expected output.'  return True  def** ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome,maxSlabTaxRate):  **assert** maxSlabAnnualIncome == 180000 **and** maxSlabTaxRate == 40 , **'Maximum tax slab does not match with expected output'  return True  def** ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome,validTaxSlabMaximumIncome):  **assert** validTaxSlabWithoutMaximumIncome **and** validTaxSlabMaximumIncome , **'Tax slab data is not valid'** |

Looking at the code it appears that following test case is not required, because it would be covered by comparing the expected output of Tax slab in subsequent test cases:

|  |
| --- |
| **assert** eachTaxSlabRow[0] >= 0 **and** eachTaxSlabRow[1] > 0 **and** eachTaxSlabRow[2] >= 0 , **'Lower income bracket and tax rate should be greater than or equal to 0. Upper income bracket should be greater than 0.'** |

So, now the code looks like:

|  |
| --- |
| **def** ValidateEachTaxSlab(eachTaxSlabRow):  **assert** isinstance(int(eachTaxSlabRow[0]),int) **and** isinstance(int(eachTaxSlabRow[1]), int) **and** isinstance(int(eachTaxSlabRow[2]), int),**'All data in Tax Slab file should be int apart from header row'  return True  def** ValidateTaxSlabWithoutMaximumIncome(entireTaxSlabWithoutMaximumIncome):  expectedOutput = [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]]  **assert** entireTaxSlabWithoutMaximumIncome == expectedOutput,**'Tax Slab does not match with expected output.'  return True  def** ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome,maxSlabTaxRate):  **assert** maxSlabAnnualIncome == 180000 **and** maxSlabTaxRate == 40 , **'Maximum tax slab does not match with expected output'  return True  def** ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome,validTaxSlabMaximumIncome):  **assert** validTaxSlabWithoutMaximumIncome **and** validTaxSlabMaximumIncome , **'Tax slab data is not valid'** |

Test results are:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest*  *================================================================================================================================================================ test session starts =================================================================================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 4 items*  *tests\unit\test\_u\_app.py .... [100%]*  *================================================================================================================================================================= 4 passed in 0.11s ==================================================================================================================================================================* |

TDD8:

Raise exception if tax slab file is not available. Test case is:

|  |
| --- |
| **def** test\_tax\_slab\_file\_exists():  **assert** os.path.isfile(os.getcwd() + **'\\GenerateMonthlyPayslip\\StaticInput\\tax\_slab.csv'**),**'Tax slab file does not exists in specified folder'** |

Refactoring this test code to unit tests:

|  |
| --- |
| **def** test\_calculate\_tax():  **assert** GetAnnualTax(60000) == 6000, **'Tax calculation not correct'  def** test\_read\_static\_tax\_file():  maxSlabAnnualIncome, maxSlabTaxRate, tax\_slab\_rate\_list = ReadTaxFile()  **assert** maxSlabAnnualIncome == 180000, **'Maximum Tax Slab annual income not correct'  assert** maxSlabTaxRate == 40, **'Maximum Tax Slab annual income not correct'  assert** tax\_slab\_rate\_list == [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]] , **'Tax Slab rate is not correct'  def** test\_calculate\_salary\_details():  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary=SalaryDetails(60000,6000)  **assert** grossMonthlySalary == 5000, **'Gross monthly income not calculated correctly'  assert** monthlyTaxPayable == 500, **'Monthly tax payable not calculated correctly'  assert** netMonthlySalary == 4500, **'Net monthly salary not calculated correctly'  def** test\_print\_employee\_monthly\_payslip():  employeeSelected = Employee(**'Mary Song'**,5000,500,4500)  employeeSelected.PrintEmployeePayslip()  expectedOutput = **'Monthly Payslip for: "Mary Song"\nGross Monthly Income: $5000\nMonthly Income Tax: $500\nNet Monthly Income: $4500'  assert** employeeSelected.PrintEmployeePayslip() == expectedOutput , **'Payslip output is not matching as expected'  def** test\_tax\_slab\_file\_exists():  **assert** os.path.isfile(os.getcwd() + **'\\GenerateMonthlyPayslip\\StaticInput\\tax\_slab.csv'**),**'Tax slab file does not exists in specified folder'** |

TDD9:

Write the system integration test. Executing following command from the command prompt:

|  |
| --- |
| python path\_to\_plutus\ GenerateMonthlyPayslip\app.py -efn "Mary Song" -gas=60000 |

Should give following result:

|  |
| --- |
| Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500 |

Added following code as integration test to the project:

|  |
| --- |
| **import** os,subprocess **def** test\_system\_integration\_message(capsys):  expectedOutput = **'Monthly Payslip for: "Mary Song"\r\nGross Monthly Income: $5000\r\nMonthly Income Tax: $500\r\nNet Monthly Income: $4500'** testCmd=**'python '** + os.getcwd() + **'\\GenerateMonthlyPayslip\\app.py -efn "Mary Song" -gas=60000'** *# os.system(testCmd)* encoding = **'utf-8'** result = subprocess.check\_output(testCmd).decode(encoding)  print(result)  captured = capsys.readouterr()  realOutput=captured.out.rstrip()  print(captured)  print(os.getcwd())  print(**'captured out is:'**)  print(realOutput)  print(**'expected out is:'**)  print(expectedOutput)  **assert** realOutput == expectedOutput , **'Expected output not observed. Integration test failed.'** |

Result of test:

|  |
| --- |
| *C:\Users\svishal\PycharmProjects\plutus>pytest -vv*  *================================================================================================================================================================ test session starts =================================================================================================================================================================*  *platform win32 -- Python 3.6.3, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 -- c:\program files (x86)\python36-32\python.exe*  *cachedir: .pytest\_cache*  *rootdir: C:\Users\svishal\PycharmProjects\plutus*  *collected 6 items*  *tests/integration/test\_i\_app.py::test\_system\_integration\_message PASSED [ 16%]*  *tests/unit/test\_u\_app.py::test\_calculate\_tax PASSED [ 33%]*  *tests/unit/test\_u\_app.py::test\_read\_static\_tax\_file PASSED [ 50%]*  *tests/unit/test\_u\_app.py::test\_calculate\_salary\_details PASSED [ 66%]*  *tests/unit/test\_u\_app.py::test\_print\_employee\_monthly\_payslip PASSED [ 83%]*  *tests/unit/test\_u\_app.py::test\_tax\_slab\_file\_exists PASSED [100%]*  *================================================================================================================================================================= 6 passed in 0.23s ==================================================================================================================================================================* |

UC1:

Parameterize config file name.

Added Linting:

First run:

|  |
| --- |
| /usr/local/bin/pylint --rcfile=setup.cfg generate\_monthly\_payslip tests  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip  generate\_monthly\_payslip/\_\_init\_\_.py:1:0: C0304: Final newline missing (missing-final-newline)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.logging\_config  generate\_monthly\_payslip/logging\_config.py:6:0: C0304: Final newline missing (missing-final-newline)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.app  generate\_monthly\_payslip/app.py:15:53: C0326: Exactly one space required after comma  def GetEmployeeMonthlyIncomeDetails(grossAnnualSalary,annualTaxToBePaid):  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:16:42: C0326: Exactly one space required after comma  return SalaryDetails(grossAnnualSalary,annualTaxToBePaid)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:19:44: C0326: Exactly one space required after comma  assert isinstance(int(grossAnnualSalary),int),'Gross Annual Salary is not int'  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:19:49: C0326: Exactly one space required after comma  assert isinstance(int(grossAnnualSalary),int),'Gross Annual Salary is not int'  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:26:29: C0326: Exactly one space required after comma  assert name.isalpha(),'Name can contain only alphabets and spaces'  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:30:24: C0326: Exactly one space required after comma  def Run(employeeFullName,grossAnnualSalaryInput):  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:34:21: C0326: Exactly one space required around assignment  grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:36:20: C0326: Exactly one space required around assignment  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:38:93: C0326: Exactly one space required after comma  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(grossAnnualSalary,annualTaxPayable)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:40:20: C0326: Exactly one space required around assignment  employeeSelected=Employee(employeeFullName,grossMonthlySalary, monthlyTaxPayable, netMonthlySalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:40:46: C0326: Exactly one space required after comma  employeeSelected=Employee(employeeFullName,grossMonthlySalary, monthlyTaxPayable, netMonthlySalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:47:46: C0326: Exactly one space required after comma  parser.add\_argument('--employee-full-name','-efn', required=True, dest='employeeFullName', help='Full name of employee')  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:48:47: C0326: Exactly one space required after comma  parser.add\_argument('--gross-annual-salary','-gas', required=True, dest='grossAnnualSalary', help='gross annual salary of employee')  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:51:29: C0326: Exactly one space required after comma  Run(args.employeeFullName,args.grossAnnualSalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:52:0: C0305: Trailing newlines (trailing-newlines)  generate\_monthly\_payslip/app.py:6:0: W0611: Unused import logging\_config (unused-import)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.calculate\_net\_monthly\_income\_details  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:1:48: C0326: Exactly one space required after comma  def CalculateNetMonthlyIncome(grossMonthlyIncome,netMonthlyTax):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:10:35: C0326: Exactly one space required after comma  def SalaryDetails(grossAnnualIncome,annualTaxToBePaid):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:11:22: C0326: Exactly one space required around assignment  grossMonthlySalary=CalculateGrossMonthlyIncome(grossAnnualIncome)  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:12:21: C0326: Exactly one space required around assignment  monthlyTaxPayable=CalculateNetMonthlyTax(annualTaxToBePaid)  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:13:20: C0326: Exactly one space required around assignment  netMonthlySalary=CalculateNetMonthlyIncome(grossMonthlySalary,monthlyTaxPayable)  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:13:65: C0326: Exactly one space required after comma  netMonthlySalary=CalculateNetMonthlyIncome(grossMonthlySalary,monthlyTaxPayable)  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:14:29: C0326: Exactly one space required after comma  return grossMonthlySalary,monthlyTaxPayable,netMonthlySalary  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:14:47: C0326: Exactly one space required after comma  return grossMonthlySalary,monthlyTaxPayable,netMonthlySalary  ^ (bad-whitespace)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.read\_tax\_slab  generate\_monthly\_payslip/read\_tax\_slab.py:5:44: C0326: Exactly one space required after comma  assert isinstance(int(eachTaxSlabRow[0]),int) and isinstance(int(eachTaxSlabRow[1]), int) and isinstance(int(eachTaxSlabRow[2]), int),'All data in Tax Slab file should be int apart from header row'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:5:137: C0326: Exactly one space required after comma  assert isinstance(int(eachTaxSlabRow[0]),int) and isinstance(int(eachTaxSlabRow[1]), int) and isinstance(int(eachTaxSlabRow[2]), int),'All data in Tax Slab file should be int apart from header row'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:10:62: C0326: Exactly one space required after comma  assert entireTaxSlabWithoutMaximumIncome == expectedOutput,'Tax Slab does not match with expected output.'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:13:52: C0326: Exactly one space required after comma  def ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome,maxSlabTaxRate):  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:14:66: C0326: No space allowed before comma  assert maxSlabAnnualIncome == 180000 and maxSlabTaxRate == 40 , 'Maximum tax slab does not match with expected output'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:17:58: C0326: Exactly one space required after comma  def ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome,validTaxSlabMaximumIncome):  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:18:74: C0326: No space allowed before comma  assert validTaxSlabWithoutMaximumIncome and validTaxSlabMaximumIncome , 'Tax slab data is not valid'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:27:22: C0326: Exactly one space required around assignment  tax\_slab\_rate\_list=[]  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:28:23: C0326: Exactly one space required around assignment  maxSlabAnnualIncome=0  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:29:18: C0326: Exactly one space required around assignment  maxSlabTaxRate=0  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:30:36: C0326: Exactly one space required around assignment  validTaxSlabWithoutMaximumIncome=False  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:31:29: C0326: Exactly one space required around assignment  validTaxSlabMaximumIncome=False  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:39:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)  generate\_monthly\_payslip/read\_tax\_slab.py:44:33: C0326: Exactly one space required around assignment  tax\_slab\_list=[int(taxSlabRow[0]),int(taxSlabRow[1]),int(taxSlabRow[2])]  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:44:53: C0326: Exactly one space required after comma  tax\_slab\_list=[int(taxSlabRow[0]),int(taxSlabRow[1]),int(taxSlabRow[2])]  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:44:72: C0326: Exactly one space required after comma  tax\_slab\_list=[int(taxSlabRow[0]),int(taxSlabRow[1]),int(taxSlabRow[2])]  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:51:80: C0326: Exactly one space required after comma  validTaxSlabMaximumIncome = ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome,maxSlabTaxRate)  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:52:58: C0326: Exactly one space required after comma  ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome,validTaxSlabMaximumIncome)  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:54:30: C0326: Exactly one space required after comma  return maxSlabAnnualIncome,maxSlabTaxRate,tax\_slab\_rate\_list  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:54:45: C0326: Exactly one space required after comma  return maxSlabAnnualIncome,maxSlabTaxRate,tax\_slab\_rate\_list  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:57:0: C0304: Final newline missing (missing-final-newline)  generate\_monthly\_payslip/read\_tax\_slab.py:46:11: W0703: Catching too general exception Exception (broad-except)  generate\_monthly\_payslip/read\_tax\_slab.py:36:12: W0612: Unused variable 'header' (unused-variable)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.calculate\_annual\_tax  generate\_monthly\_payslip/calculate\_annual\_tax.py:8:16: C0326: No space allowed after bracket  def GetAnnualTax( annualIncome ):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_annual\_tax.py:8:31: C0326: No space allowed before bracket  def GetAnnualTax( annualIncome ):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_annual\_tax.py:11:20: C0326: Exactly one space required after comma  maxTaxSlabIncome,maxTaxSlabRate, taxSlabRanges = ReadTaxFile()  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_annual\_tax.py:2:4: C0414: Import alias does not rename original package (useless-import-alias)  generate\_monthly\_payslip/calculate\_annual\_tax.py:4:4: C0414: Import alias does not rename original package (useless-import-alias)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.print\_employee\_monthly\_payslip  generate\_monthly\_payslip/print\_employee\_monthly\_payslip.py:9:28: C0326: Exactly one space required around assignment  payslipOutputMessage='Monthly Payslip for: "' + self.\_\_employeeFullName + '"\n'  ^ (bad-whitespace)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.tests.unit.test\_u\_app  tests/unit/test\_u\_app.py:2:67: C0326: No space allowed before comma  from ...generate\_monthly\_payslip.read\_tax\_slab import ReadTaxFile , GetCsvFilenameFromConfigFile  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:14:110: C0326: No space allowed before comma  assert tax\_slab\_rate\_list == [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]] , 'Tax Slab rate is not correct'  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:17:59: C0326: Exactly one space required around assignment  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary=SalaryDetails(60000,6000)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:17:79: C0326: Exactly one space required after comma  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary=SalaryDetails(60000,6000)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:23:43: C0326: Exactly one space required after comma  employeeSelected = Employee('Mary Song',5000,500,4500)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:23:48: C0326: Exactly one space required after comma  employeeSelected = Employee('Mary Song',5000,500,4500)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:23:52: C0326: Exactly one space required after comma  employeeSelected = Employee('Mary Song',5000,500,4500)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:26:69: C0326: No space allowed before comma  assert employeeSelected.PrintEmployeePayslip() == expectedOutput , 'Payslip output is not matching as expected'  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:29:0: C0304: Final newline missing (missing-final-newline)  tests/unit/test\_u\_app.py:29:94: C0326: Exactly one space required after comma  assert os.path.isfile(os.getcwd() + '/generate\_monthly\_payslip/static\_input/tax\_slab.csv'),'Tax slab file does not exists in specified folder'  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:1:0: C0414: Import alias does not rename original package (useless-import-alias)  tests/unit/test\_u\_app.py:2:0: W0611: Unused GetCsvFilenameFromConfigFile imported from generate\_monthly\_payslip.read\_tax\_slab (unused-import)  tests/unit/test\_u\_app.py:5:0: C0411: standard import "import os" should be placed before "from ...generate\_monthly\_payslip.calculate\_annual\_tax import GetAnnualTax as GetAnnualTax" (wrong-import-order)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.tests.integration.test\_i\_app  tests/integration/test\_i\_app.py:1:9: C0326: Exactly one space required after comma  import os,subprocess  ^ (bad-whitespace)  tests/integration/test\_i\_app.py:10:11: C0326: Exactly one space required around assignment  testCmd=["python3", os.getcwd() + '/generate\_monthly\_payslip/app.py', "-efn", "Mary Song", "-gas", "60000"]  ^ (bad-whitespace)  tests/integration/test\_i\_app.py:16:14: C0326: Exactly one space required around assignment  realOutput=captured.out.rstrip()  ^ (bad-whitespace)  tests/integration/test\_i\_app.py:23:0: C0304: Final newline missing (missing-final-newline)  tests/integration/test\_i\_app.py:23:40: C0326: No space allowed before comma  assert realOutput == expectedOutput , 'Expected output not observed. Integration test failed.'  ^ (bad-whitespace)  tests/integration/test\_i\_app.py:1:0: C0410: Multiple imports on one line (os, subprocess) (multiple-imports)  -----------------------------------  Your code has been rated at 5.70/10 |

2nd run:

|  |
| --- |
| /usr/local/bin/pylint --rcfile=setup.cfg generate\_monthly\_payslip tests  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.app  generate\_monthly\_payslip/app.py:30:24: C0326: Exactly one space required after comma  def Run(employeeFullName,grossAnnualSalaryInput):  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:34:21: C0326: Exactly one space required around assignment  grossAnnualSalary=ValidateConvertAnnualSalary(grossAnnualSalaryInput)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:36:20: C0326: Exactly one space required around assignment  annualTaxPayable=GetEmployeeAnnualTax(grossAnnualSalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:51:29: C0326: Exactly one space required after comma  Run(args.employeeFullName,args.grossAnnualSalary)  ^ (bad-whitespace)  generate\_monthly\_payslip/app.py:52:0: C0305: Trailing newlines (trailing-newlines)  generate\_monthly\_payslip/app.py:6:0: W0611: Unused import logging\_config (unused-import)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.calculate\_net\_monthly\_income\_details  generate\_monthly\_payslip/calculate\_net\_monthly\_income\_details.py:13:67: C0326: Exactly one space required after comma  netMonthlySalary = CalculateNetMonthlyIncome(grossMonthlySalary,monthlyTaxPayable)  ^ (bad-whitespace)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.read\_tax\_slab  generate\_monthly\_payslip/read\_tax\_slab.py:5:44: C0326: Exactly one space required after comma  assert isinstance(int(eachTaxSlabRow[0]),int) and isinstance(int(eachTaxSlabRow[1]), int) and isinstance(int(eachTaxSlabRow[2]), int), 'All data in Tax Slab file should be int apart from header row'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:14:66: C0326: No space allowed before comma  assert maxSlabAnnualIncome == 180000 and maxSlabTaxRate == 40 , 'Maximum tax slab does not match with expected output'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:18:74: C0326: No space allowed before comma  assert validTaxSlabWithoutMaximumIncome and validTaxSlabMaximumIncome , 'Tax slab data is not valid'  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:39:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)  generate\_monthly\_payslip/read\_tax\_slab.py:51:80: C0326: Exactly one space required after comma  validTaxSlabMaximumIncome = ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome,maxSlabTaxRate)  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:52:58: C0326: Exactly one space required after comma  ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome,validTaxSlabMaximumIncome)  ^ (bad-whitespace)  generate\_monthly\_payslip/read\_tax\_slab.py:57:0: C0304: Final newline missing (missing-final-newline)  generate\_monthly\_payslip/read\_tax\_slab.py:46:11: W0703: Catching too general exception Exception (broad-except)  generate\_monthly\_payslip/read\_tax\_slab.py:36:12: W0612: Unused variable 'header' (unused-variable)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.calculate\_annual\_tax  generate\_monthly\_payslip/calculate\_annual\_tax.py:8:16: C0326: No space allowed after bracket  def GetAnnualTax( annualIncome ):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_annual\_tax.py:8:31: C0326: No space allowed before bracket  def GetAnnualTax( annualIncome ):  ^ (bad-whitespace)  generate\_monthly\_payslip/calculate\_annual\_tax.py:2:4: C0414: Import alias does not rename original package (useless-import-alias)  generate\_monthly\_payslip/calculate\_annual\_tax.py:4:4: C0414: Import alias does not rename original package (useless-import-alias)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.tests.unit.test\_u\_app  tests/unit/test\_u\_app.py:17:81: C0326: Exactly one space required after comma  grossMonthlySalary, monthlyTaxPayable, netMonthlySalary = SalaryDetails(60000,6000)  ^ (bad-whitespace)  tests/unit/test\_u\_app.py:29:0: C0304: Final newline missing (missing-final-newline)  tests/unit/test\_u\_app.py:1:0: C0414: Import alias does not rename original package (useless-import-alias)  tests/unit/test\_u\_app.py:5:0: C0411: standard import "import os" should be placed before "from ...generate\_monthly\_payslip.calculate\_annual\_tax import GetAnnualTax as GetAnnualTax" (wrong-import-order)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.tests.integration.test\_i\_app  tests/integration/test\_i\_app.py:1:9: C0326: Exactly one space required after comma  import os,subprocess  ^ (bad-whitespace)  tests/integration/test\_i\_app.py:1:0: C0410: Multiple imports on one line (os, subprocess) (multiple-imports)  ------------------------------------------------------------------  Your code has been rated at 8.49/10 (previous run: 5.70/10, +2.79) |

3rd run:

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.app  generate\_monthly\_payslip/app.py:6:0: W0611: Unused import logging\_config (unused-import)  \*\*\*\*\*\*\*\*\*\*\*\*\* Module plutus.generate\_monthly\_payslip.read\_tax\_slab  generate\_monthly\_payslip/read\_tax\_slab.py:46:11: W0703: Catching too general exception Exception (broad-except)  generate\_monthly\_payslip/read\_tax\_slab.py:36:12: W0612: Unused variable 'header' (unused-variable)  ------------------------------------------------------------------  Your code has been rated at 9.83/10 (previous run: 8.49/10, +1.34) |

Bug1:

When using flask or other code outside the folder hosting read\_tax\_slab.py, it throws following error:

|  |
| --- |
| *INFO:generate\_monthly\_payslip.app:Welcome to the world of "Plutus". Starting payslip generator application.*  *Traceback (most recent call last):*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/app.py", line 49, in <module>*  *Run(args.employeeFullName, args.grossAnnualSalary)*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/app.py", line 34, in Run*  *annualTaxPayable = GetEmployeeAnnualTax(grossAnnualSalary)*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/app.py", line 13, in GetEmployeeAnnualTax*  *return GetAnnualTax(grossAnnualSalary)*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/calculate\_annual\_tax.py", line 8, in GetAnnualTax*  *maxTaxSlabIncome, maxTaxSlabRate, taxSlabRanges = ReadTaxFile()*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/read\_tax\_slab.py", line 51, in ReadTaxFile*  *validTaxSlabWithoutMaximumIncome = ValidateTaxSlabWithoutMaximumIncome(tax\_slab\_rate\_list)*  *File "/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/read\_tax\_slab.py", line 10, in ValidateTaxSlabWithoutMaximumIncome*  *assert entireTaxSlabWithoutMaximumIncome == expectedOutput, 'Tax Slab does not match with expected output.'*  *AssertionError: Tax Slab does not match with expected output.* |

Analysis:

We are getting into this issue, because we are trying to get the absolute path of the tax\_slab.csv file. The line which is faulting is:

|  |
| --- |
| def ReadTaxFile():  inputStaticTaxSlabFile = os.path.abspath('static\_input') + '/tax\_slab.csv' |

Changing it to Python 3.6 pythonic way should fix it. Pseudo code could be:

|  |
| --- |
| **from** pathlib **import** Path  inputStaticTaxSlabFile = Path(\_\_file\_\_).parent / **"static\_input/tax\_slab.csv"** |

Refactoring the code:

|  |
| --- |
| import csv  import os  from pathlib import Path  def ValidateEachTaxSlab(eachTaxSlabRow):  assert isinstance(int(eachTaxSlabRow[0]), int) and isinstance(int(eachTaxSlabRow[1]), int) and isinstance(int(eachTaxSlabRow[2]), int), 'All data in Tax Slab file should be int apart from header row'  return True  def ValidateTaxSlabWithoutMaximumIncome(entireTaxSlabWithoutMaximumIncome):  expectedOutput = [[0, 20000, 0], [20000, 40000, 10], [40000, 80000, 20], [80000, 180000, 30]]  assert entireTaxSlabWithoutMaximumIncome == expectedOutput, 'Tax Slab does not match with expected output.'  return True  def ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome, maxSlabTaxRate):  assert maxSlabAnnualIncome == 180000 and maxSlabTaxRate == 40, 'Maximum tax slab does not match with expected output'  return True  def ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome, validTaxSlabMaximumIncome):  assert validTaxSlabWithoutMaximumIncome and validTaxSlabMaximumIncome, 'Tax slab data is not valid'  def GetCsvFilenameFromConfigFile():  configFileWithLocation = "../setup\_config"  with open(configFileWithLocation, "r") as configFile:  return configFile.readline().split('=')[1]  def ReadTaxFile():  inputStaticTaxSlabFile = Path(\_\_file\_\_).parent / "static\_input/tax\_slab.csv"  print (inputStaticTaxSlabFile)  tax\_slab\_rate\_list = []  maxSlabAnnualIncome = 0  maxSlabTaxRate = 0  validTaxSlabWithoutMaximumIncome = False  validTaxSlabMaximumIncome = False  try:  with open(inputStaticTaxSlabFile) as csvfile:  csv\_file = csv.reader(csvfile)  header = next(csv\_file)  taxSlab = csv\_file  for taxSlabRow in taxSlab:  if taxSlabRow[1] == '-1':  maxSlabAnnualIncome = int(taxSlabRow[0])  maxSlabTaxRate = int(taxSlabRow[2])  else:  ValidateEachTaxSlab(taxSlabRow)  tax\_slab\_list = [int(taxSlabRow[0]), int(taxSlabRow[1]), int(taxSlabRow[2])]  tax\_slab\_rate\_list.append(tax\_slab\_list)  except Exception as exception:  print("File not accessible")  print(format(exception))  validTaxSlabWithoutMaximumIncome = ValidateTaxSlabWithoutMaximumIncome(tax\_slab\_rate\_list)  validTaxSlabMaximumIncome = ValidateTaxSlabMaximumIncome(maxSlabAnnualIncome, maxSlabTaxRate)  ValidateEntireTaxSlab(validTaxSlabWithoutMaximumIncome, validTaxSlabMaximumIncome)  return maxSlabAnnualIncome, maxSlabTaxRate, tax\_slab\_rate\_list  if \_\_name\_\_ == '\_\_main\_\_':  ReadTaxFile() |

Reran unit tests:

Integration test failed after this fix:

|  |
| --- |
| > assert realOutput == expectedOutput, 'Expected output not observed. Integration test failed.'  E AssertionError: Expected output not observed. Integration test failed.  E assert "['python3', ...Income: $4500" == "['python3', ...Income: $4500"  E ['python3', '/home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/app.py', '-efn', 'Mary Song', '-gas', '60000']  E - /home/svishal/git-repos/generate\_payslip/plutus/generate\_monthly\_payslip/static\_input/tax\_slab.csv  E Monthly Payslip for: "Mary Song"  E Gross Monthly Income: $5000  E Monthly Income Tax: $500  E Net Monthly Income: $4500  tests/integration/test\_i\_app.py:24: AssertionError |

It seems that the tax file name is getting printed as highlighted above. This is occurring because we added a step to print the tax\_slab.csv file path to troubleshoot! So, offending code is:

|  |
| --- |
| def ReadTaxFile():  inputStaticTaxSlabFile = Path(\_\_file\_\_).parent / "static\_input/tax\_slab.csv"  print (inputStaticTaxSlabFile)  tax\_slab\_rate\_list = []  maxSlabAnnualIncome = 0  maxSlabTaxRate = 0  validTaxSlabWithoutMaximumIncome = False  validTaxSlabMaximumIncome = False |

Removing the print statement should fix it. Refactored code.

Now the integration tests succeeds but api is not giving desired output:

|  |
| --- |
| *tests/unit/test\_u\_app.py::test\_calculate\_tax PASSED [ 12%]*  *tests/unit/test\_u\_app.py::test\_read\_static\_tax\_file PASSED [ 25%]*  *tests/unit/test\_u\_app.py::test\_calculate\_salary\_details PASSED [ 37%]*  *tests/unit/test\_u\_app.py::test\_print\_employee\_monthly\_payslip PASSED [ 50%]*  *tests/unit/test\_u\_app.py::test\_tax\_slab\_file\_exists PASSED [ 62%]*  *tests/unit/test\_u\_app.py::test\_if\_api\_is\_running PASSED [ 75%]*  *tests/unit/test\_u\_app.py::test\_if\_api\_gives\_expected\_output FAILED [ 87%]*  *tests/integration/test\_i\_app.py::test\_system\_integration\_message PASSED [100%]* |

The error message is:

|  |
| --- |
| > assert response.content == expectedOutput, 'API is not giving expected output'  E AssertionError: API is not giving expected output  E assert b'Monthly Pay...come: $4500\n' == 'Monthly Pays...Income: $4500'  E -b'Monthly Payslip for: "Mary Song"\nGross Monthly Income: $5000\nMonthly Income Tax: $500\nNet Monthly Income: $4500\n'  E +'Monthly Payslip for: "Mary Song"\nGross Monthly Income: $5000\nMonthly Income Tax: $500\nNet Monthly Income: $4500'  tests/unit/test\_u\_app.py:42: AssertionError |

So, we are comparing byte string with normal string. The offending code is:

|  |
| --- |
| def test\_if\_api\_gives\_expected\_output():  expectedOutput = '''Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500'''  response = requests.get('http://127.0.0.1:5000/api/v1/resources/employee?fullName=Mary%20Song&grossAnnualIncome=60000')  assert response.content == expectedOutput, 'API is not giving expected output' |

So, response.content should be encoded to compare. Refactored the code to following:

|  |
| --- |
| def test\_if\_api\_gives\_expected\_output():  expectedOutput = '''Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500'''  response = requests.get('http://127.0.0.1:5000/api/v1/resources/employee?fullName=Mary%20Song&grossAnnualIncome=60000')  encoding = 'utf-8'  assert response.content.decode(encoding) == expectedOutput, 'API is not giving expected output' |

Running tests again, it still failed:

|  |
| --- |
| *> assert response.content.decode(encoding) == expectedOutput, 'API is not giving expected output'*  *E AssertionError: API is not giving expected output*  *E assert 'Monthly Pays...come: $4500\n' == 'Monthly Pays...Income: $4500'*  *E Monthly Payslip for: "Mary Song"*  *E Gross Monthly Income: $5000*  *E Monthly Income Tax: $500*  *E - Net Monthly Income: $4500*  *E ? -*  *E + Net Monthly Income: $4500* |

So, it is occurring because expected output does not have a new line character. Changing test case to use valid expected output should fix it.

So, now the test looks like:

|  |
| --- |
| def test\_if\_api\_gives\_expected\_output():  expectedOutput = '''Monthly Payslip for: "Mary Song"  Gross Monthly Income: $5000  Monthly Income Tax: $500  Net Monthly Income: $4500  '''  response = requests.get('http://127.0.0.1:5000/api/v1/resources/employee?fullName=Mary%20Song&grossAnnualIncome=60000')  encoding = 'utf-8'  assert response.content.decode(encoding) == expectedOutput, 'API is not giving expected output' |

Running tests again, and voila it succeeds:

|  |
| --- |
| *================================================================================ test session starts =================================================================================*  *platform linux -- Python 3.6.0, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 -- /usr/local/bin/python3.6*  *cachedir: .pytest\_cache*  *rootdir: /home/svishal/git-repos/generate\_payslip/plutus, inifile: setup.cfg, testpaths: tests/unit, tests/integration*  *collected 8 items*  *tests/unit/test\_u\_app.py::test\_calculate\_tax PASSED [ 12%]*  *tests/unit/test\_u\_app.py::test\_read\_static\_tax\_file PASSED [ 25%]*  *tests/unit/test\_u\_app.py::test\_calculate\_salary\_details PASSED [ 37%]*  *tests/unit/test\_u\_app.py::test\_print\_employee\_monthly\_payslip PASSED [ 50%]*  *tests/unit/test\_u\_app.py::test\_tax\_slab\_file\_exists PASSED [ 62%]*  *tests/unit/test\_u\_app.py::test\_if\_api\_is\_running PASSED [ 75%]*  *tests/unit/test\_u\_app.py::test\_if\_api\_gives\_expected\_output PASSED [ 87%]*  *tests/integration/test\_i\_app.py::test\_system\_integration\_message PASSED [100%]*  *================================================================================= 8 passed in 0.23s ==================================================================================* |