The shop CSV should hold the initial cash value for the shop.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| @dataclass  class Shop:  cash: float = 0.0  stock: List[ProductStock] = field(default\_factory=list)  def create\_and\_stock\_shop():  *# create shop and assign values*  s = Shop()  with open('../stock.csv') as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  s.cash = float(first\_row[0])  for row in csv\_reader:  p = Product(row[0], float(row[1]))  ps = ProductStock(p, float(row[2]))  s.stock.append(ps)  *#print(ps)*  return s | class Shop:    def \_\_init\_\_(self, path):  self.stock = []  with open(path) as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  self.cash = float(first\_row[0])  for row in csv\_reader:  p = Product(row[0], float(row[1]))  ps = ProductStock(p, float(row[2]))  self.stock.append(ps)  def find\_product(self, product\_name):  for item in self.stock:  if str(product\_name) == str(item.name()):  return 1  return 0  def find\_product\_qty(self, product\_name):  for item in self.stock:  if str(product\_name) == str(item.name()):  return item.get\_available\_qty()  return 0    def find\_product\_price(self, product\_name):  for item in self.stock:  if str(product\_name) == str(item.name()):  return item.unit\_price()  return 0    def update\_cash(self, cash):  self.cash = cash    def \_\_repr\_\_(self):  str = ""  str += f'Shop has {self.cash} in cash\n'  str += f'The Shop has the following stock which will be cheked against your order\n'  str += f'\n'  for item in self.stock:  str += f"{item}\n"  return str |
| What makes this a data class is the [@dataclass decorator](https://realpython.com/primer-on-python-decorators/) just above the class definition. Beneath the class Position: line, you simply list the fields you want in your data class. The : notation used for the fields is using a new feature in Python 3.6 called [variable annotations](https://www.python.org/dev/peps/pep-0526/).  https://realpython.com/python-data-classes/ | <https://www.toptal.com/python/python-class-attributes-an-overly-thorough-guide>  In here I use normal class function .So when we creating this class object it requires a csv file path.  s = Shop("../stock.csv") |

Read in customer orders from a CSV file.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| elif int(answer) == 2:  print("---------------------------------------------------")  customer\_csv\_path = all\_csv\_files[int(answer)-2]  print(f'you selected {customer\_csv\_path} as a current customer data')  print('')  c = read\_customer(str(customer\_csv\_path))  print\_customer(c)  print('')  print("---------------------------------------------------")  def read\_customer(file\_path):  with open(file\_path) as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  c = Customer(first\_row[0], float(first\_row[1]))  for row in csv\_reader:  name = row[0]  quantity = int(row[1])  price = float(find\_product\_price(name))  p = Product(name, price)  ps = ProductStock(p, quantity)  c.shopping\_list.append(ps)  return c  ef print\_customer(c):  print(f'CUSTOMER NAME: {c.name} \nCUSTOMER BUDGET: {c.budget}')  total = 0.0  out\_of\_stock = []  active\_product\_list = []  for item in c.shopping\_list:  *# check weather this product in stock*  check\_val = find\_product\_price(item.product.name)  if check\_val == 0:  print('')  print(f'Error no {item.product.name} in Shop stock.List item ignored')  else:  print\_product(item.product)  print('')  print(f'{c.name} ORDERS {item.quantity} OF ABOVE PRODUCT')  if int(find\_product\_qty(item.product.name)) < int(item.quantity):  data = {  "product\_name" : item.product.name,  "availble\_qty" : find\_product\_qty(item.product.name),  "need\_qty": item.quantity  }  out\_of\_stock.append(data)  cost = item.quantity \* item.product.price  buying\_data = {  "product" : item.product.name,  "qty" : item.quantity,  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)    total = total + cost  print(f'The cost to {c.name} will be €{str(round(cost, 2))}')  print('--------------------------------------------------')  print(f'The total cost to {c.name} will be €{total}') | print("---------------------------------------------------")  customer\_csv\_path = all\_csv\_files[int(answer)-2]  print(f'you selected {customer\_csv\_path} as a current customer data')  print('')  c = Customer(customer\_csv\_path)  print(c)  print('')  print("---------------------------------------------------")  class Customer:  def \_\_init\_\_(self, path):  self.shopping\_list = []  with open(path) as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  self.name = first\_row[0]  self.budget = float(first\_row[1])  for row in csv\_reader:  name = row[0]  quantity = float(row[1])  price = s.find\_product\_price(name)  p = Product(name, price)  ps = ProductStock(p, quantity)  self.shopping\_list.append(ps)    def calculate\_costs(self, price\_list):  for shop\_item in price\_list:  for list\_item in self.shopping\_list:  if (list\_item.name() == shop\_item.name()):  list\_item.product.price = shop\_item.unit\_price()    def order\_cost(self):  cost = 0    for list\_item in self.shopping\_list:  cost += list\_item.cost()    return cost    def \_\_repr\_\_(self):    str = f"CUSTOMER NAME : {self.name}\n"  str += f"CUSTOMER BUDGET: {self.budget}\n"  total = 0.0  out\_of\_stock = []  active\_product\_list = []  for item in self.shopping\_list:  check\_val = s.find\_product\_price(item.name())  if check\_val == 0:  str += '\n'  str += f'Error no {item.name()} in Shop stock.List item ignored'  else:  str += f'\nPRODUCT NAME: {item.name()} \nPRODUCT PRICE: {item.unit\_price()}'  str += f'\n{self.name} ORDERS {item.get\_available\_qty()} OF ABOVE PRODUCT'  if int(s.find\_product\_qty(item.name())) < int(item.get\_available\_qty()):  data = {  "product\_name" : item.name(),  "availble\_qty" : int(s.find\_product\_qty(item.name())),  "need\_qty": int(item.get\_available\_qty())  }  out\_of\_stock.append(data)  cost = int(item.get\_available\_qty()) \* float(item.unit\_price())  buying\_data = {  "product" : item.name(),  "qty" : item.get\_available\_qty(),  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)  total = total + cost  str += f'\nThe cost to {self.name} will be €{round(cost, 2)}'    str += '\n--------------------------------------------------'  str += f'\nThe total cost to {self.name} will be €{total}'  if len(out\_of\_stock) == 0:  if float(c.budget) >= float(total):  str += '\n--------------------------------------------------'  str += '\nSUCCESS !'  str += f"\n{self.name}'s budget is {self.budget}"  str += f"\n{self.name} has enough money"  str += f"\ntotal of {total} will be deducted from {self.name}'s budget"  *# updating the stock*  for stock in s.stock:  *# print(stock)*  for product in active\_product\_list:  if stock.name() == product['product']:  qty = int(stock.quantity) - int(product['qty'])  stock.update\_qty(qty)    self.budget = float(self.budget) - float(total)  shop\_total = float(s.cash) + float(total)  s.update\_cash(shop\_total)  str += f"\n{self.name}'s budget is {self.budget}"  str += f"\ntotal of {total} added to shop"  str += f"\nShop has {s.cash}"  else:  str += '\n--------------------------------------------------'  str += '\nFAIL !'  str += f"\n{self.name}'s budget is {self.budget}"  str += f'\nThe total cost of all items is ...! {float(total)}'  str += f"\n{self.name} does not have enough money"  str += f"\ntotal of 0 will be deducted from {self.name}'s budget"  str += f"\n{self.name}'s budget is {self.budget}"  str += f"\ntotal of 0 added to shop"  str += f"\nShop has {s.cash} in cash"  else:  for out\_stock in out\_of\_stock:  str += f"\nNot enough {out\_stock['product\_name']} in stock"  str += f"\n{out\_stock['availble\_qty']} {out\_stock['product\_name']} in stock"  str += f"\n{self.name} want {out\_stock['need\_qty']} {out\_stock['product\_name']}"  str += "\nPlease revise order and upload again!"    return str |
| Both procedural and oop code block using same function to read csv file and append items into a list.  https://realpython.com/python-csv/ | To check the available quantity , price and name ,in here I use the shop class, because I wrote simple function inside of the shop class to retrieve above details.  https://stackoverflow.com/questions/14086830/python-calling-method-in-class/14086857 |

That file should include all the products they wish to buy and in what quantity.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| total = 0.0  out\_of\_stock = []  active\_product\_list = []  for item in c.shopping\_list:  *# check weather this product in stock*  check\_val = find\_product\_price(item.product.name)  if check\_val == 0:  print('')  print(f'Error no {item.product.name} in Shop stock.List item ignored')  else:  print\_product(item.product)  print('')  print(f'{c.name} ORDERS {item.quantity} OF ABOVE PRODUCT')  if int(find\_product\_qty(item.product.name)) < int(item.quantity):  data = {  "product\_name" : item.product.name,  "availble\_qty" : find\_product\_qty(item.product.name),  "need\_qty": item.quantity  }  out\_of\_stock.append(data)  cost = item.quantity \* item.product.price  buying\_data = {  "product" : item.product.name,  "qty" : item.quantity,  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)    total = total + cost  print(f'The cost to {c.name} will be €{str(round(cost, 2))}')  print('--------------------------------------------------')  print(f'The total cost to {c.name} will be €{total}') | for item in self.shopping\_list:  check\_val = s.find\_product\_price(item.name())  if check\_val == 0:  str += '\n'  str += f'Error no {item.name()} in Shop stock.List item ignored'  else:  str += f'\nPRODUCT NAME: {item.name()} \nPRODUCT PRICE: {item.unit\_price()}'  str += f'\n{self.name} ORDERS {item.get\_available\_qty()} OF ABOVE PRODUCT'  if int(s.find\_product\_qty(item.name())) < int(item.get\_available\_qty()):  data = {  "product\_name" : item.name(),  "availble\_qty" : int(s.find\_product\_qty(item.name())),  "need\_qty": int(item.get\_available\_qty())  }  out\_of\_stock.append(data)  cost = int(item.get\_available\_qty()) \* float(item.unit\_price())  buying\_data = {  "product" : item.name(),  "qty" : item.get\_available\_qty(),  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)  total = total + cost  str += f'\nThe cost to {self.name} will be €{round(cost, 2)}' |
| In this code block we can see its using foreach loop to check customer shopping list with shop items list. In this point it will trigger out those items are available or not. | In oop code block also check with the foreach loop with customer shop items.  https://www.geeksforgeeks.org/iterate-over-a-list-in-python/ |

It should also include their name and their budget.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| for item in s.stock:  print\_product(item.product)  print(f'The Shop has {int(item.quantity)} of the above') | def \_\_repr\_\_(self):  str = ""  str += f'Shop has {self.cash} in cash\n'  str += f'The Shop has the following stock which will be cheked against your order\n'  str += f'\n'  for item in self.stock:  str += f"{item}\n"  return str |
| In here we can access key and value in the shop list items. | In oop we call this list in Shop class, In here \_\_repr\_\_ method is doing amazing job. When you create a object of shop class and print the reference using it.This object will return the string from above method.  S = Shop(‘csv\_file\_path’)  Print(s) Example The following code shows how \_\_repr\_\_() is used.  class Point:  def \_\_init\_\_(self, x, y):  self.x, self.y = x, y     def \_\_repr\_\_(self):       return 'Point(x=%s, y=%s)' % (self.x, self.y)  p = Point(3, 4)  print p Output This gives the output  Point(x=3, y=4) |
|  |  |

The shop must be able to process the orders of the customer.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| for item in c.shopping\_list:  *# check weather this product in stock*  check\_val = find\_product\_price(item.product.name)  if check\_val == 0:  print('')  print(f'Error no {item.product.name} in Shop stock.List item ignored')  else:  print\_product(item.product)  print('')  print(f'{c.name} ORDERS {item.quantity} OF ABOVE PRODUCT')  if int(find\_product\_qty(item.product.name)) < int(item.quantity):  data = {  "product\_name" : item.product.name,  "availble\_qty" : find\_product\_qty(item.product.name),  "need\_qty": item.quantity  }  out\_of\_stock.append(data)  cost = item.quantity \* item.product.price  buying\_data = {  "product" : item.product.name,  "qty" : item.quantity,  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)    total = total + cost  print(f'The cost to {c.name} will be €{str(round(cost, 2))}') | for item in self.shopping\_list:  check\_val = s.find\_product\_price(item.name())  if check\_val == 0:  str += '\n'  str += f'Error no {item.name()} in Shop stock.List item ignored'  else:  str += f'\nPRODUCT NAME: {item.name()} \nPRODUCT PRICE: {item.unit\_price()}'  str += f'\n{self.name} ORDERS {item.get\_available\_qty()} OF ABOVE PRODUCT'  if int(s.find\_product\_qty(item.name())) < int(item.get\_available\_qty()):  data = {  "product\_name" : item.name(),  "availble\_qty" : int(s.find\_product\_qty(item.name())),  "need\_qty": int(item.get\_available\_qty())  }  out\_of\_stock.append(data)  cost = int(item.get\_available\_qty()) \* float(item.unit\_price())  buying\_data = {  "product" : item.name(),  "qty" : item.get\_available\_qty(),  "sub\_tot" : cost  }  active\_product\_list.append(buying\_data)  total = total + cost  str += f'\nThe cost to {self.name} will be €{round(cost, 2)}' |
| Within this code block check the out of stock with list items values within if code block. If its true then append that customer ordered item to out\_of\_stock [] list. | Otherwise it will append to active\_product\_list for further information gathering ,like stock update, shop price updated and update customer budget as well. |

Update the cash in the shop based on money received.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| *# updating the stock*  for stock in s.stock:  *# print(stock)*  for product in active\_product\_list:  if stock.product.name == product['product']:  *# print('product name is there')*  stock.quantity = float(stock.quantity) - float(product['qty'])    c.budget = float(c.budget) - float(total)  s.cash = float(s.cash) + float(total)  print(f"John's budget is {c.budget}")  print(f"total of {total} added to shop")  print(f"Shop has {s.cash}") | *#update the stock*  for stock in s.stock:  *# print(stock)*  for product in active\_product\_list:  if stock.name() == product['product']:  *# print('product name is there')*  qty = int(stock.quantity) - int(product['qty'])  stock.update\_qty(qty)  shop\_total = float(s.cash) + float(total)  s.update\_cash(shop\_total)  custom\_budget = float(custom\_budget) - float(total)  *# print(s.stock[0].product.name, s.stock[0].quantity)* |
| In here get the reference of shop and update it cash flow and stock and also customer budget as well | <https://stackoverflow.com/questions>  /20923411/updating-class-variable-within-a-instance-method |

It is important that the state of the shop be consistent.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| s = create\_and\_stock\_shop() | s = Shop("../stock.csv") |
| In here its calling to the direct methods | In this block Shop object will be create |

You should create customer test files (CSVs) which cannot be completed by the shop e.g. customer wants 400 loaves of bread but the shop only has 20, or the customer wants 2 cans of coke but can only afford 1.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| if int(find\_product\_qty(item.product.name)) < int(item.quantity):  data = {  "product\_name" : item.product.name,  "availble\_qty" : find\_product\_qty(item.product.name),  "need\_qty": item.quantity  }  out\_of\_stock.append(data) | if int(s.find\_product\_qty(item.name())) < int(item.get\_available\_qty()):  data = {  "product\_name" : item.name(),  "availble\_qty" : int(s.find\_product\_qty(item.name())),  "need\_qty": int(item.get\_available\_qty())  }  out\_of\_stock.append(data) |
| In here Simply we can check it calling find\_product\_qty it will return available qty in stock for check the availability | In here that get\_available\_qty() function calling by shop instance |

– Know whether or not the shop can fill an order.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| *# check weather this product in stock*  check\_val = find\_product\_price(item.product.name)  if check\_val == 0:  print('')  print(f'Error no {item.product.name} in Shop stock.List item ignored') | check\_val = s.find\_product\_price(item.name())  if check\_val == 0:  str += '\n'  str += f'Error no {item.name()} in Shop stock.List item ignored' |
| https://stackoverflow.com/questions/7571635/fastest-way-to-check-if-a-value-exists-in-a-list | https://stackoverflow.com/questions/7571635/fastest-way-to-check-if-a-value-exists-in-a-list |

\* Thrown an appropriate error.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| else:  print('--------------------------------------------------')  print('FAIL !')  print(f"{c.name}'s budget is {c.budget}")  print(f'The total cost of all items is ...! {float(total)}')  print(f"{c.name} does not have enough money")  print(f"total of 0 will be deducted from {c.name}'s budget")  print(f"John's budget is {c.budget}")  print(f"total of 0 added to shop")  print(f"Shop has {s.cash} in cash")  else:  for out\_stock in out\_of\_stock:  print(f"Not enough {out\_stock['product\_name']} in stock")  print(f"{out\_stock['availble\_qty']} {out\_stock['product\_name']} in stock")  print(f"{c.name} want {out\_stock['need\_qty']} {out\_stock['product\_name']}")  print("Please revise order and upload again!") | else:  str += '\n--------------------------------------------------'  str += '\nFAIL !'  str += f"\n{self.name}'s budget is {self.budget}"  str += f'\nThe total cost of all items is ...! {float(total)}'  str += f"\n{self.name} does not have enough money"  str += f"\ntotal of 0 will be deducted from {self.name}'s budget"  str += f"\n{self.name}'s budget is {self.budget}"  str += f"\ntotal of 0 added to shop"  str += f"\nShop has {s.cash} in cash"  else:  for out\_stock in out\_of\_stock:  str += f"\nNot enough {out\_stock['product\_name']} in stock"  str += f"\n{out\_stock['availble\_qty']} {out\_stock['product\_name']} in stock"  str += f"\n{self.name} want {out\_stock['need\_qty']} {out\_stock['product\_name']}"  str += "\nPlease revise order and upload again!" |
| This error is handle with main infinite while loop with try catch in both procedural and oop  .to the requirement I use if else statement to return custom error messages. | <https://www.w3schools.com/python/>  python\_try\_except.asp |

Operate in a live mode, where the user can enter a product by name, specify a quantity, and pay for it. The user should be able to buy many products in this way.

|  |  |
| --- | --- |
| PROCEDURAL CODE | OOP CODE |
| if int(answer) == 1:  custom\_active = True  print("---------------------------------------------------")  print('Live mode')  custom\_budget = input('Type your budget : ')  active\_product\_list = []  while custom\_active:  custom\_product = input('Enter a product name you would like? Eg: Coke Can, Bread, Spaghetti, Tomato Sauce, Big Bags : ')  if not find\_product(custom\_product):  print('This product : {} not in stock '.format(custom\_product))  else:  print('Entered Product is {}'.format(custom\_product))    custom\_count = input('How many do you want? : ')  if int(find\_product\_qty(custom\_product)) >= int(custom\_count):  custom\_sub\_total = float(find\_product\_price(custom\_product)) \* int(custom\_count)  print('The cost to you will be {} '.format(round(custom\_sub\_total, 2)))  data = {  "product" : custom\_product,  "qty" : custom\_count,  "sub\_tot" : custom\_sub\_total  }  active\_product\_list.append(data)  else:  print('')  print('FAIL ! ')  print(f'Not enough {custom\_product} in stock ')  print(f'{int(find\_product\_qty(custom\_product))} {custom\_product} in stock')  print(f'You want {custom\_count} {custom\_product}')  print('Please revise order and try again!')  close\_tag = input('Would you like another product? Y/N ')  if close\_tag == 'N':  custom\_active = False    print('Summary')  if len(active\_product\_list) > 0:  total = 0  for product in active\_product\_list:  total = total + float(product['sub\_tot'])    if float(total) > float(custom\_budget):  print('FAIL! ')  print(f"Your budget is ...")  print(f'The total cost of all items is ...! {total}')  print("You do not have enough money")  print(f"Total of 0 will be deducted from your budget")  print(f"Your budget is {custom\_budget}")  print(f"Total of 0 added to shop")  print(f"Shop has {round(float(s.cash), 2)} in cash")  else:  print('SUCCESS! ')  print(f"Your budget is {custom\_budget}")  print("You have enough money")  print(f"Total of {round(total,2)} will be deducted from your budget")  print(f"Total of {round(total,2)} added to shop")  *#update the stock*  for stock in s.stock:  *# print(stock)*  for product in active\_product\_list:  if stock.product.name == product['product']:  *# print('product name is there')*  stock.quantity = float(stock.quantity) - float(product['qty'])  shop\_total = float(s.cash) + float(total)  s.cash = shop\_total  custom\_budget = float(custom\_budget) - float(total)  print(s.stock[0].product.name, s.stock[0].quantity)    print(f"Shop has {round(shop\_total, 2)} in cash")  else:  print('something went wrong')    print("---------------------------------------------------") | if int(answer) == 1:  custom\_active = True  print("---------------------------------------------------")  print('Live mode')  custom\_budget = input('Type your budget : ')  active\_product\_list = []  while custom\_active:  custom\_product = input('Enter a product name you would like? Eg: Coke Can, Bread, Spaghetti, Tomato Sauce, Big Bags : ')  if not s.find\_product(custom\_product):  print('This product : {} not in stock '.format(custom\_product))  else:  print('Entered Product is {}'.format(custom\_product))  custom\_count = input('How many do you want? : ')  if int(s.find\_product\_qty(custom\_product)) >= int(custom\_count):  custom\_sub\_total = float(s.find\_product\_price(custom\_product)) \* int(custom\_count)  print('The cost to you will be {} '.format(round(custom\_sub\_total, 2)))  data = {  "product" : custom\_product,  "qty" : custom\_count,  "sub\_tot" : custom\_sub\_total  }  active\_product\_list.append(data)  else:  print('')  print('FAIL ! ')  print(f'Not enough {custom\_product} in stock ')  print(f'{int(s.find\_product\_qty(custom\_product))} {custom\_product} in stock')  print(f'You want {custom\_count} {custom\_product}')  print('Please revise order and try again!')  close\_tag = input('Would you like another product? Y/N ')  if close\_tag == 'N':  custom\_active = False  print('Summary')    if len(active\_product\_list) > 0:  total = 0  for product in active\_product\_list:  total = total + float(product['sub\_tot'])    if float(total) > float(custom\_budget):  print('FAIL! ')  print(f"Your budget is ...")  print(f'The total cost of all items is ...! {total}')  print("You do not have enough money")  print(f"Total of 0 will be deducted from your budget")  print(f"Your budget is {custom\_budget}")  print(f"Total of 0 added to shop")  print(f"Shop has {round(float(s.cash), 2)} in cash")  else:  print('SUCCESS! ')  print(f"Your budget is {custom\_budget}")  print("You have enough money")  print(f"Total of {round(total,2)} will be deducted from your budget")  print(f"Total of {round(total,2)} added to shop")  *#update the stock*  for stock in s.stock:  *# print(stock)*  for product in active\_product\_list:  if stock.name() == product['product']:  *# print('product name is there')*  qty = int(stock.quantity) - int(product['qty'])  stock.update\_qty(qty)  shop\_total = float(s.cash) + float(total)  s.update\_cash(shop\_total)  custom\_budget = float(custom\_budget) - float(total)  *# print(s.stock[0].product.name, s.stock[0].quantity)*    *# print(f"Shop has {round(shop\_total, 2)} in cash")*  else:  print('something went wrong')  print("---------------------------------------------------") |
| In here also live mode work inside of infinite live mode. This will terminate by predefined flag. If the user need to exit from live mode that variable name update to FALSE.  To get the command line data using of ‘input()’  In python | So that clarification same as in here.  https://www.geeksforgeeks.org/taking-input-from-console-in-python/ |