

```
from DBHelper import DBHelper
from helper_functions import *
#This file will contain all API functions calls exposed to outside world for users to use
```

```
# function about Product
def create_product(products, code, name, units):
    result = products.create(code, name, units)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Product Create Success.')
    return result #send result for caller program to use
```

```
def read_product(products, code):
    result = products.read(code) #returns tuple of (error dict, data dict)
    if result[0]['Is Error']: #in case error
        print(result[0]['Error Message'])
    else:
        print(result[1])
    return result #send result for caller program to use
```

```
def update_product(products, code, newName, newUnits):
    result = products.update(code, newName, newUnits) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Product Update Success.')
    return result #send result for caller program to use
```

```
def delete_product(products, code):
    result = products.delete(code)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Product Delete Success.')
    return result #send result for caller program to use
```

```
def report_list_products(products):
    result = products.dump()
    #printDictInCSVFormat(result, ('Code',), ('Name', 'Units'))
    print (result)
    return result #send result for caller program to use
```

```
# function about Customer
def create_customer(customers, customerCode, customerName, address, creditLimit, country):
    result = customers.create(customerCode, customerName, address, creditLimit, country)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Customer Create Success.')
    return result #send result for caller program to use
```

```
def read_customer(customers, customerCode):
    result = customers.read(customerCode) #returns tuple of (error dict, data dict)
    if result[0]['Is Error']: #in case error
        print(result[0]['Error Message'])
    else:
        print(result[1])
    return result #send result for caller program to use
```

```
def update_customer(customers, customerCode, newCustomerName, newAddress, newCreditLimit, newCountry):
    result = customers.update(customerCode, newCustomerName, newAddress, newCreditLimit, newCountry) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Customer Update Success.')
    return result #send result for caller program to use
```

```
def delete_customer(customers, customerCode):
    result = customers.delete(customerCode)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Customer Delete Success.')
    return result #send result for caller program to use
```

```
def report_list_all_customers(customers):
    result = customers.dump()
    printDictInCSVFormat(result, ('Customer Code',), ('Name', 'Address','Credit Limit', 'Country'))
    return result #send result for caller program to use
```

```
# function about Invoice
def create_invoice(invoices, invoiceNo, invoiceDate, customerCode, dueDate, invoiceLineTuplesList):
    if invoiceDate == None:
        invoiceDate = 'null'
    else:
        invoiceDate = "" + invoiceDate + ""
    if dueDate == None:
        dueDate = 'null'
    else:
        dueDate = "" + dueDate + ""
    result = invoices.create(invoiceNo, invoiceDate, customerCode, dueDate, invoiceLineTuplesList)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Invoice Create Success.')
    return result #send result for caller program to use
```

```
def read_invoice(invoices, invoiceNo):
    result = invoices.read(invoiceNo) #returns tuple of (error dict, data dict)
    if result[0]['Is Error']: #in case error
        print(result[0]['Error Message'])
    else:
        print(result[1])
    return result #send result for caller program to use
```

```
def update_invoice(invoices, invoiceNo, newInvoiceDate, newCustomerCode, newDueDate, newInvoiceLineTuplesList):
    if newInvoiceDate == None:
        newInvoiceDate = 'null'
    else:
        newInvoiceDate = "" + newInvoiceDate + ""
    if newDueDate == None:
        newDueDate = 'null'
    else:
        newDueDate = "" + newDueDate + ""
    result = invoices.update(invoiceNo, newInvoiceDate, newCustomerCode, newDueDate, newInvoiceLineTuplesList) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Invoice Update Success.')
    return result #send result for caller program to use
```

```
def delete_invoice(invoices, invoiceNo):
    result = invoices.delete(invoiceNo)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Invoice Delete Success.')
    return result #send result for caller program to use
```

```
def update_invoice_line(invoices, invoiceNo, productCode, newQuantity, newUnitPrice):
    result = invoices.update_invoice_line(invoiceNo, productCode, newQuantity, newUnitPrice) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Invoice Line Item Update Success.')
    return result #send result for caller program to use
```

```
def delete_invoice_line(invoices, invoiceNo, productCode):
    result = invoices.delete_invoice_line(invoiceNo, productCode) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Invoice Line Item Delete Success.')
    return result #send result for caller program to use
```

```
def report_list_all_invoices(invoices, customers, products):
    # Will dump all invoices data and return 1 dictionary as a result (with header and line item joined).
    # Please show the customer name and product name also.
    # A helper function such as def print_tabular_dictionary(tabularDictionary) can then be called to print this in a tabular (table-like) form with column headings and data.
```

```
db = DBHelper()
data, columns = db.fetch ('SELECT i.invoice_no as "Invoice No", i.date as "Date" '
    ' , i.customer_code as "Customer Code", c.name as "Customer Name" '
    ' , i.due_date as "Due Date", i.total as "Total", i.vat as "VAT", i.amount_due as "Amount Due" '
    ' , ili.product_code as "Product Code", p.name as "Product Name" '
    ' , ili.quantity as "Quantity", ili.unit_price as "Unit Price", ili.extended_price as "Extended Price" '
    ' FROM invoice i JOIN customer c ON i.customer_code = c.customer_code '
    ' JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
    ' JOIN product p ON ili.product_code = p.code '
    ''
)

#print (result)
result = row_as_dict(data, columns)
printDictInCSVFormat(result, ('Invoice No'),), ('Date', 'Customer Code', 'Customer Name','Due Date','Total','VAT','Amount Due'
    , 'Product Code', 'Product Name', 'Quantity', 'Unit Price', 'Extended Price'))

return result #send result for caller program to use
```

```
def report_products_sold(invoices, products, dateStart, dateEnd):
    db = DBHelper()
    data, columns = db.fetch ('SELECT p.code as "Code", ili.product_code as "Product Code", p.name as "Product Name" '
        ' , SUM(ili.quantity) as "Total Quantity Sold", SUM(ili.extended_price) as "Total Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' GROUP BY p.code, ili.product_code, p.name ')
    result = row_as_dict(data, columns)
    data, columns = db.fetch ('SELECT 0 as "Footer", SUM(ili.extended_price) as "Total Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' ')
    result2 = row_as_dict(data, columns)

    printDictInCSVFormat(result, (None), ('Product Code','Product Name', 'Total Quantity Sold', 'Total Value Sold'))
    printDictInCSVFormat(result2, (None), ('Total Value Sold',))
    return result, result2
```

```
def report_customer_products_sold_list(invoices, products, customers, dateStart, dateEnd):
    db = DBHelper()
    data, columns = db.fetch ('SELECT i.customer_code, c.customer_code as "Customer Code", c.name as "Customer Name" '
        ' , ili.product_code as "Product Code", p.name as "Product Name" '
        ' , i.invoice_no as "Invoice No" '
        ' , SUM(ili.quantity) as "Quantity Sold", SUM(ili.extended_price) as "Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN customer c ON i.customer_code = c.customer_code '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' GROUP BY i.customer_code, c.customer_code, c.name, i.invoice_no, ili.product_code, p.name ')
    result = row_as_dict(data, columns)
    data, columns = db.fetch ('SELECT 0 as "Footer", SUM(ili.quantity) as "Quantity Sold", SUM(ili.extended_price) as "Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN customer c ON i.customer_code = c.customer_code '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' ')
    result2 = row_as_dict(data, columns)

    printDictInCSVFormat(result, (None), ('Customer Code','Customer Name', 'Product Code', 'Product Name', 'Invoice No', 'Quantity Sold', 'Value Sold'))
    printDictInCSVFormat(result2, (None), ('Quantity Sold','Value Sold'))
    return result.values(), result2
```

```
def report_customer_products_sold_total(invoices, products, customers, dateStart, dateEnd):
    # Will return 2 dictionaries:
    # 1) a dictionary as list customers and the total number and value of products sold to them in the given date range in this format: Customer Code, Customer Name, Product Code, Product Name, Total Quantity Sold, Total Value Sold. Here (customer code, product code) will be unique.
    # And 2) a second footer dictionary showing: t the end also show the sum of Total Quantity Sold, sum of Total Value Sold.
    db = DBHelper()
    data, columns = db.fetch ('SELECT i.customer_code, c.customer_code as "Customer Code", c.name as "Customer Name" '
        ' , ili.product_code as "Product Code", p.name as "Product Name" '
        ' , SUM(ili.quantity) as "Total Quantity Sold", SUM(ili.extended_price) as "Total Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN customer c ON i.customer_code = c.customer_code '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' GROUP BY i.customer_code, c.customer_code, c.name, i.invoice_no, ili.product_code, p.name ')
    result = row_as_dict(data, columns)
    data, columns = db.fetch ('SELECT 0 as "Footer", SUM(ili.quantity) as "Total Quantity Sold", SUM(ili.extended_price) as "Total Value Sold" '
        ' FROM invoice i JOIN invoice_line_item ili ON i.invoice_no = ili.invoice_no '
        ' JOIN customer c ON i.customer_code = c.customer_code '
        ' JOIN product p ON ili.product_code = p.code '
        ' WHERE i.date between \'' + dateStart + '\' and \'' + dateEnd + '\' '
        ' ')
    result2 = row_as_dict(data, columns)

    printDictInCSVFormat(result, (None), ('Customer Code','Customer Name', 'Product Code', 'Product Name', 'Total Quantity Sold', 'Total Value Sold'))
    printDictInCSVFormat(result2, (None), ('Total Quantity Sold','Total Value Sold'))
    return result.values(), result2
```

```
# function about Payment method
def create_PaymentMethod(PaymentMethods, paymentMethodCode, paymentMethodName):
    result = PaymentMethods.create(paymentMethodCode, paymentMethodName)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Payment Method Create Success.')
    return result #send result for caller program to use
```

```
def read_PaymentMethod(PaymentMethods, paymentMethodCode):
    result = PaymentMethods.read(paymentMethodCode) #returns tuple of (error dict, data dict)
    if result[0]['Is Error']: #in case error
        print(result[0]['Error Message'])
    else:
        print(result[1])
    return result #send result for caller program to use
```

```
def update_PaymentMethod(PaymentMethods, paymentMethodCode, newPaymentMethodName):
    result = PaymentMethods.update(paymentMethodCode, newPaymentMethodName) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Payment Method Update Success.')
    return result #send result for caller program to use
```

```
def delete_PaymentMethod(PaymentMethods, paymentMethodCode):
    result = PaymentMethods.delete(paymentMethodCode)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Payment Method Delete Success.')
    return result #send result for caller program to use
```

```
def report_list_payment_methods(PaymentMethods):
    result = PaymentMethods.dump()
    print(result)
    return result #send result for caller program to use
```

```
#function about Receipt
def create_receipt(receipts, receiptNo, receiptDate, customerCode, paymenMethod, paymenReference, remark, receiptLinelItemList):
    if receiptDate == None:
        receiptDate = 'null'
    else:
        receiptDate = "" + receiptDate + ""
    result = receipts.create(receiptNo, receiptDate, customerCode, paymenMethod, paymenReference, remark, receiptLinelItemList)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Receipts Create Success.')
    return result #send result for caller program to use
```

```
def read_receipt(receipts, receiptNo):
    result = receipts.read(receiptNo) #returns tuple of (error dict, data dict)
    if result[0]['Is Error']: #in case error
        print(result[0]['Error Message'])
    else:
        print(result[1])
    return result #send result for caller program to use
```

```
def update_receipt(receipts, receiptNo, newReceiptDate, newCustomerCode, newPaymenMethod, newPaymenReference, newRemark, newReceiptLinelItemList):
    if newReceiptDate == None:
        newReceiptDate = 'null'
    else:
        newReceiptDate = "" + newReceiptDate + ""

    result = receipts.update(receiptNo, newReceiptDate, newCustomerCode, newPaymenMethod, newPaymenReference, newRemark, newReceiptLinelItemList) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Receipt Update Success.')
    return result #send result for caller program to use
```

```
def delete_receipt(receipts, receiptNo):
    result = receipts.delete(receiptNo)#returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Receipt Delete Success.')
    return result #send result for caller program to use
```

```
def report_list_all_receipts(receipts, invoices, customers):
    db = DBHelper()
    data, columns = db.fetch ('SELECT r.receipt_no as "Receipt No", r.receipt_date as "Receipt Date", r.customer_code as "Customer Code", c.name as "Customer Name" '
        ', r.payment_method as "Payment Method", r.payment_reference as "Payment Reference", r.remark as "Remark", rli.amount_paid_here As "Total Receipt"'
        ', i.invoice_no AS "Invoice No", i.Date AS "Invoice Date", i.amount_due AS "Invoice Amount Due" ,rli.amount_paid_here AS "Invoice Amount Received"'
        'FROM receipt r JOIN receipt_line_item rli ON rli.receipt_no = r.receipt_no '
        'JOIN customer c ON c.customer_code = r.customer_code '
        'JOIN invoice i ON i.invoice_no = rli.invoice_no')

    #print (result)
    result = row_as_dict(data, columns)
    #printDictInCSVFormat(result, ('Receipt No'),, ('Receipt Date', 'Customer Code', 'Customer Name', 'Payment Method','Payment Reference','Remark','Total Receipt','Invoice No','Invoice Date','Invoice Amount Due', 'Invoice Amount Received'))
    #send result for caller program to use
    print(str(columns)[1:-1])
    for i in data:
        s = []
        for j in i:
            s.append(str(j))
        print(', '.join(s))
    return result
```

```
def update_receipt_line(receipts, receiptNo, invoiceNo, newAmountPaid):
    result = receipts.update_receipt_line(receiptNo, invoiceNo, newAmountPaid) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Receipt Line Item Update Success.')
    return result #send result for caller program to use
```

```
def delete_receipt_line(receipts, receiptNo, invoiceNo):
    result = receipts.delete_receipt_line(receiptNo, invoiceNo) #returns error dictionary
    if result['Is Error']: #if error
        print(result['Error Message'])
    else:
        print('Receipt Line Item Delete Success.')
    return result #send result for caller program to use
```

```
def report_unpaid_invoices(invoices,customers,receipts):
    db = DBHelper()
    data, columns = db.fetch (' select i.invoice_no AS "Invoice No" , i.date AS "Invoice Date" , c.name AS "Customer Name" , i.amount_due AS "Invoice Amount Due" ,
                                'sum(rli.amount_paid_here) AS "Invoice Amount Received" , (i.amount_due - sum(rli.amount_paid_here)) AS "Unpaid"'
                                ' FROM receipt r JOIN receipt_line_item rli ON rli.receipt_no = r.receipt_no '
                                ' JOIN invoice i ON i.invoice_no = rli.invoice_no '
                                ' JOIN customer c ON c.customer_code = i.customer_code '
                                ' Group by i.invoice_no, c.name , i.amount_due;')

    #print (result)
    result = row_as_dict(data, columns)
```

```
db = DBHelper()
data, columns = db.fetch (' select 0 as "Footer" , count(unpaid) as "Number of invoices not paid" , sum(unpaid) as "Total unpaid" , sum("Amount Paid Here") as "Total Receipt"'
                            ' from (SELECT rli."invoice_no" as "Invoice No", i.date as "Invoice Date", c.name as "Customer Name" , '
                            ' i."amount_due" as "Amount Received", SUM(rli.amount_paid_here) as "Amount Paid Here", '
                            ' (i.amount_due - sum(rli.amount_paid_here)) as "unpaid" '
                            ' FROM receipt r JOIN receipt_line_item rli ON r."receipt_no" = rli."receipt_no"'
                            ' JOIN invoice i ON i."invoice_no" = rli."invoice_no" '
                            ' JOIN customer c ON c."customer_code" = i."customer_code" '
                            ' GROUP BY rli."invoice_no" ,i."date" , c."name",i."amount_due") as total_un_re;')
```

```
#print (result)
result2 = row_as_dict(data, columns)
```

```
printDictInCSVFormat(result, ('Invoice No'), ('Invoice Date', 'Customer Name', 'Invoice Amount Due', 'Invoice Amount Received', 'Unpaid'))
printDictInCSVFormat(result2, (None), ('Number of invoices not paid', 'Total unpaid', 'Total Receipt'))
```

```
return result, result2
```