

EXPERIMENT - 4

FRASON FRANCIS / 201903020 / SE-IT

Aim: Creating User-defined modules/packages and importing them in a program.

Theory:

A module is a piece of software that has a specific functionality. A Python module is a file that contains Python code.

For example, when building a shopping cart application, you can have one module for calculating prices and another module for managing items in the cart. Each module is a separate Python source code file.

A module has a name specified by the filename without the .py extension. For example, if you have a file called pricing.py, the module name is pricing.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: jkfrason
"""
# Writing the Python module
# pricing.py

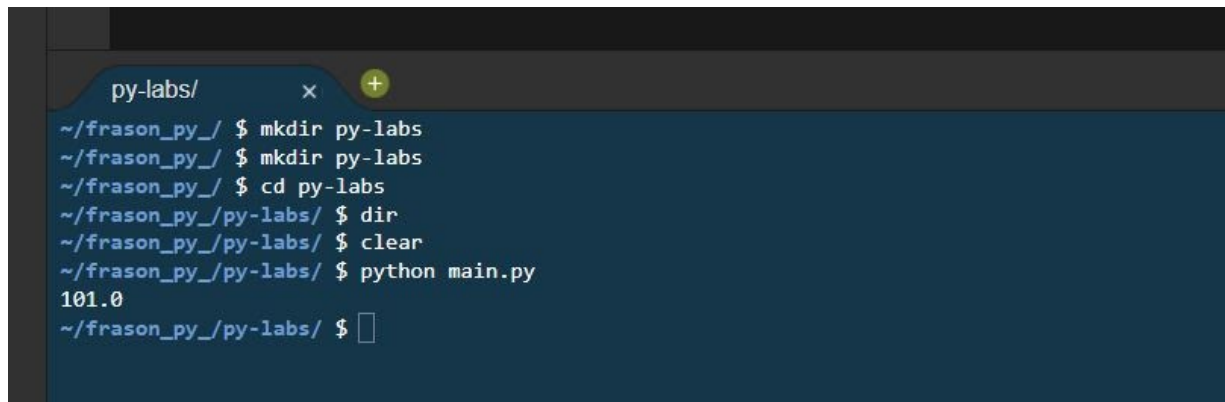
def get_net_price(price, tax_rate, discount=0):
    return price * (1 + tax_rate) * (1-discount)

def get_tax(price, tax_rate=0):
    return price * tax_rate
```

```
#main.py
import pricing
```

```
net_price = pricing.get_net_price(  
    price=100,  
    tax_rate=0.01  
)  
  
print(net_price)
```

Output:

A terminal window with a dark blue background and white text. The window title is 'py-labs/'. The terminal shows the following commands and output:

```
~/frason_py/ $ mkdir py-labs  
~/frason_py/ $ mkdir py-labs  
~/frason_py/ $ cd py-labs  
~/frason_py/py-labs/ $ dir  
~/frason_py/py-labs/ $ clear  
~/frason_py/py-labs/ $ python main.py  
101.0  
~/frason_py/py-labs/ $
```

Aim: Creating user defined multithreaded application with thread synchronization and deadlocks.

Theory:

A **Lock** is an object that acts as a permit. The lock will be assigned to only one thread at a time. The other threads will wait for the lock owner to complete its task and return it. The Lock mechanism will be possible to control the competition between the various threads, ensuring that each one of them performs its activities without the unwanted interference of the other threads.

There might be the case that a thread that has received a lock, for some reason, never returns it. In this case the program would remain completely blocked.

To avoid these cases, the locks of the threading module have been designed to work also as a context manager, within a statement with. With this system, the lock will still be released automatically when the block with ends. In this way it will not be

necessary to call the **acquire()** and **release()** functions, but everything will be managed by the context manager.

The use of locks is an optimal solution to avoid race condition problems, but sometimes, especially if used outside of the construct, they can create some problem

In that case it will be necessary to manage the locks with the **acquire()** and **release()** methods, and once the lock has been acquired by a thread, all the other threads will have to wait for its release before proceeding further.

This, if not well managed, can lead to **deadlocks**. Therefore for some reason the thread with the lock will never release hence the program will remain locked.

Imagine an online payment checkout, some tasks that need to be completed during checkout are the following:

1. Verifying Payment/Card Details
2. Sending Confirmation email or shipping details
3. Loading a thank you page or redirecting back to main website

Code:

```
"""
@author: jkfrason

Aim: Creating user defined multithreaded application with thread synchronization and deadlocks
"""
#Application on Multithreading sync with deadlock on a typical web page scenario

import threading #module
import time #sync with time for deadlock

class myThread1(threading.Thread): #creating subclass from main class thread
    def __init__(self, threadId, name, count): #method
        threading.Thread.__init__(self) #parent-constructor method called
        self.threadId = threadId
        self.name = name
        self.count = count #In our example we will count on numbers so we use count
```

```

def run(self):      #To run the thread (method)
    print("Starting: " + self.name + "\n")
    threadLock.acquire() #lock the thread so that no other thread run unless this thread
finishes
    print_time(self.name, 1,self.count) #giving the delay of 1'sec
    threadLock.release() #To release the lock of the thread
    print("Exiting: " + self.name + "\n") #Done with thread

class myThread2(threading.Thread):
    def __init__(self, threadId, name, count):
        threading.Thread.__init__(self)
        self.threadId = threadId
        self.name = name
        self.count = count

    def run(self):
        print("Starting: " + self.name + "\n")
        threadLock.acquire()
        threadLock.release()
        print_time(self.name, 1,self.count)
        print("Exiting: " + self.name + "\n")

def print_time(name, delay, count):
    while count:
        time.sleep(delay) #delay/sleep by
        print ("%s: %s %s" % (name, time.ctime(time.time()), count) + "\n") #time.ctime giving
current time
        count -= 1 #for counting down

threadLock = threading.Lock()

thread1 = myThread1(1, "Payment", 5) #waiting for 5 sec
thread2 = myThread2(2, "Sending Email", 10) #waiting for 10 sec, will be executed after
payment
thread3 = myThread2(3, "Loading Page", 3) #waiting for 3 sec will be executed after the sending
email

#Execution of thread

thread1.start()
thread2.start()
thread3.start()
thread1.join() #To wait fot the thread to finish before exiting the program
thread2.join()
thread3.join()
print("Done main thread")

```

Output:

Console 5/A X

```
In [2]: runcell(0, 'C:/Users/jkfra/Desktop/Py-Labs/Exp4_frason.py')
```

Starting: Payment

Starting: Sending Email

Starting: Loading Page

Sending Email: Thu Apr 22 19:10:25 2021 10

Payment: Thu Apr 22 19:10:25 2021 5

Sending Email: Thu Apr 22 19:10:26 2021 9

Payment: Thu Apr 22 19:10:26 2021 4

Sending Email: Thu Apr 22 19:10:27 2021 8

Payment: Thu Apr 22 19:10:27 2021 3

Sending Email: Thu Apr 22 19:10:28 2021 7

Payment: Thu Apr 22 19:10:28 2021 2

Sending Email: Thu Apr 22 19:10:29 2021 6

Payment: Thu Apr 22 19:10:29 2021 1

Exiting: Payment

Sending Email: Thu Apr 22 19:10:30 2021 5

Loading Page: Thu Apr 22 19:10:30 2021 3

Sending Email: Thu Apr 22 19:10:31 2021 4

Loading Page: Thu Apr 22 19:10:31 2021 2

Sending Email: Thu Apr 22 19:10:32 2021 3

Loading Page: Thu Apr 22 19:10:32 2021 1

Exiting: Loading Page

IPython console Plots

```
Sending Email: Thu Apr 22 19:10:32 2021 3
```

```
Loading Page: Thu Apr 22 19:10:32 2021 1
```

```
Exiting: Loading Page
```

```
Sending Email: Thu Apr 22 19:10:33 2021 2
```

```
Sending Email: Thu Apr 22 19:10:34 2021 1
```

```
Exiting: Sending Email
```

```
Done main thread
```

```
In [3]:
```

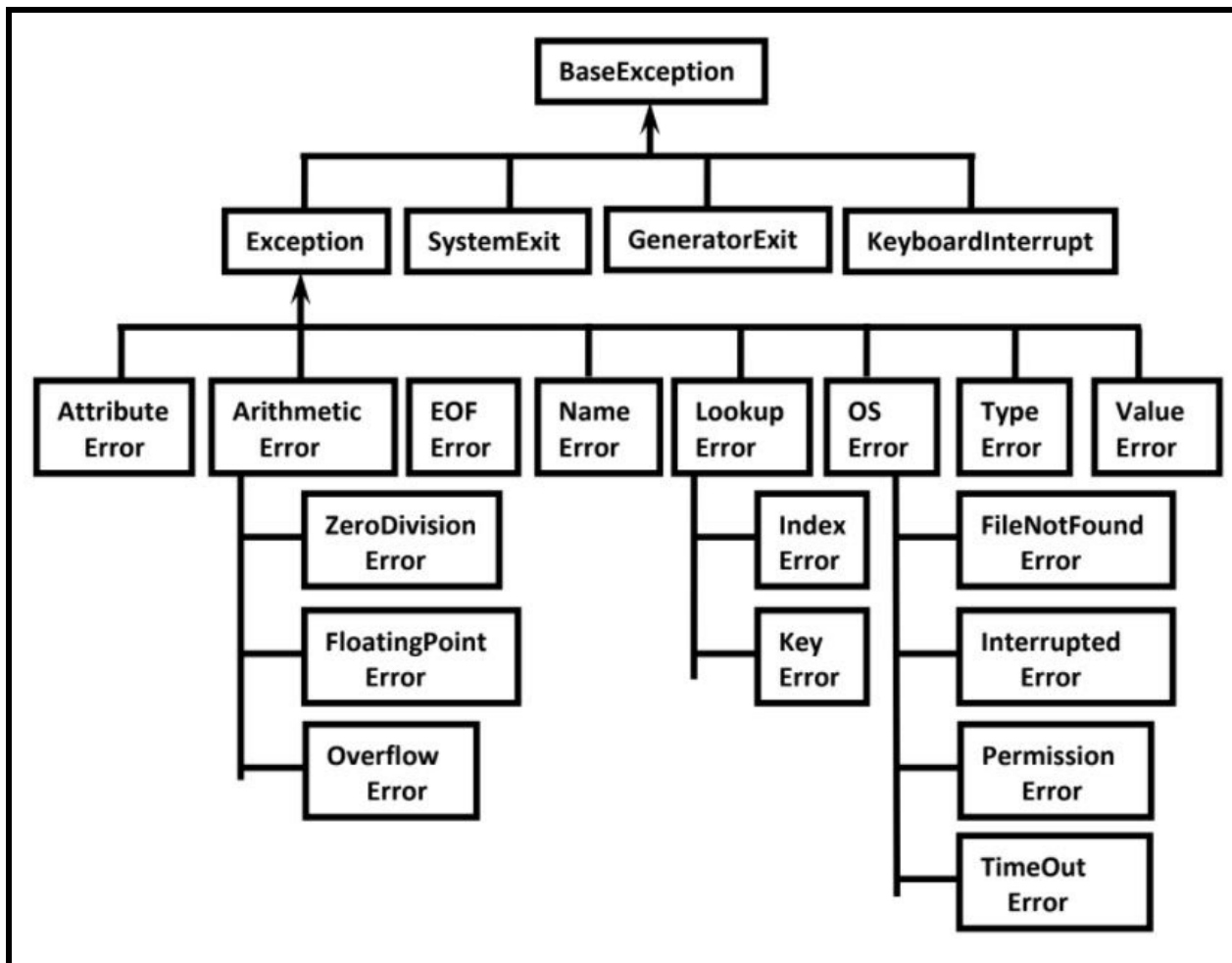
Aim: Creating a menu driven application which should cover all the exceptions in python.

Theory:

An unwanted and unexpected event that disturbs normal flow of program is called Exception.

Types of Exception:

- ZeroDivisionError
- TypeError
- ValueError
- FileNotFoundError
- EOFError
- SleepingError
- TyrePuncturedError



- **IndexError** – You will get this error when an index is not found in a sequence. For instance, accessing the 6th index when the length of the list is only five(5).
- **IndentationError** – Happens when indentation is not specified correctly.
- **ValueError** – Occurs when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values.
- **IOError** – Developers often encounter this error when an input/output operation fails.
- **Arithmetic Error** – Occurs when numeric calculations fail.
- **Floating-point Error** – Happens when a floating-point calculation fails.
- **Assertion Error** – Occurs when there is assert statement failure.
- **Overflow Error** – Developers get this error if the result of an arithmetic operation is too large and becomes machine unreadable.

- **Type Error** – Happens when an incorrect type of function or operation is applied to an object.

Restaurant Billing menu driven using Python with exception handling

Code:

```
# -*- coding: utf-8 -*-
"""
@author: jkfrason
"""
import sys

print("NOTHING BUT CHICKEN")
print ("MENU CARD")

lst_choice = []
lst_qty = []
dict_menu = {1:'Naan', 2:'Chicken Tikka', 3:'Butter Chicken', 4:'Ice Cream'}
dict_price = {1:10, 2:200, 3:500, 4:20}

while(1):
    print('1: Naan Rs.10/piece \n2: Chicken Tikka Rs.200/plate \n3: Butter Chicken Rs.500/plate \n4: Ice Cream Rs. 20/serving')
    try:
        a = input('Do you want to order?(y/n)')

    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        continue

    if(a == 'y' or a == 'Y'):
        try:
            choice = int(input('Enter choice'))

        except:
            print("Oops!",sys.exc_info()[0],"occured.")
            continue

        if(choice >= 1 and choice <= 4):
            try:
                qty = int(input('How much qty do you need:'))

            except:
```



```
        print("Oops!",sys.exc_info()[0],"occured.")
        continue

    if(qty >= 1):
        lst_choice.append(choice)
        lst_qty.append(qty)
        print('\n')

    else:
        print('Wrong Input \n')
        continue

    else:
        print('Wrong Input \n')
        continue

elif(a == 'n' or a == 'N'):
    print("\n\nBILL:")
    total = 0
    length = len(lst_choice)

    for i in range(0, length):
        val = lst_choice[i]
        print(dict_menu[val])
        print('Qty: ', lst_qty[i])
        print('Rs.', dict_price[val], 'per unit')
        print('\n')
        total = total + (dict_price[val] * lst_qty[i])

    print('total = Rs.', total)
    print('Thank you')
    break

else:
    print("Wrong input \n")
    continue
```

Output:

```
In [2]: runcell(0, 'C:/Users/jkfra/Desktop/Py-Labs/EXP-4C.py')
NOTHING BUT CHICKEN
MENU CARD
1: Naan Rs.10/piece
2: Chicken Tikka Rs.200/plate
3: Butter Chicken Rs.500/plate
4: Ice Cream Rs. 20/serving

Do you want to order?(y/n)Y

Enter choice3

How much qty do you need:2

1: Naan Rs.10/piece
2: Chicken Tikka Rs.200/plate
3: Butter Chicken Rs.500/plate
4: Ice Cream Rs. 20/serving

Do you want to order?(y/n)3
Wrong input

1: Naan Rs.10/piece
2: Chicken Tikka Rs.200/plate
3: Butter Chicken Rs.500/plate
4: Ice Cream Rs. 20/serving

Do you want to order?(y/n)n

BILL:
Butter Chicken
Qty: 2
Rs. 500 per unit

total = Rs. 1000
Thank you

In [3]: |
```

IPython console

Plots