exception

June 19, 2023

[ ]:
```
#Q1
```

[ ]:
```
'exception are the type of module which can be called for exception handling␣
 ↪cases'
'exception are occured durning the excecution of programme which distrub the␣
 ↪normal flow of instruction'
```

[ ]:
```
1.'exception are runtime error that error occur during the excecution of the␣
 ↪programme'
2.'exception are caused due to the invalid entered input , unexcepted␣
 ↪conditions
```

[ ]:
```
1.'syntax error occur normally when the coder does not follow the instruction␣
 ↪of the particular python language
2.'for example syntax error include missing coloum or calling the name of the␣
 ↪module with out defineing it etc.'
```

[ ]:
```
#Q2
```

[ ]:
```
'exception is not handled it leads to an unhandling exception when it occurs␣
 ↪the programme gets out of the loop and presents a error message is displayed'
```

[16]:
```
import logging
def divide_number(a,b):
    logging.basicConfig(filename='divide.txt',level=logging.
 ↪INFO,format='%(asctime)s-%(levelname)s-%(message)s')
    try :
        with open('data.txt','w') as f:
            result = a/b
            return result
    except ZeroDivision as e:
        logging.info('error occured:%s',str(e))
    try:
        result = divide_number(10,0)
        logging.info('Result:%s',result)
    except ZeroDivisionError:
        logging.info('Error: cannot be divide by zero')
```

```
    result = divide_number(10,0)
    logging.info('Result:%s',result)
```

[17]:
```
#Q3
```

[ ]:
```
except exception as e :
```

[2]:
```
import logging

logging.basicConfig(level=logging.ERROR, format='%(levelname)s: %(message)s')

def divide_numbers(dividend, divisor):
    try:
        result = dividend / divisor
        logging.info("Division result: %s", result)
    except ZeroDivisionError:
        logging.error("Cannot divide by zero!")
```

[ ]:
```
#Q4
```

[3]:
```
import logging
logging.basicConfig(filename='try.txt', level=logging.
 ↪INFO,format='%l(evelname)s: %(message)s')
def is_prime(number):
    try:
        if number<2:
            raise valueerror('number must be greater than or equal to 2')
        for i in range(2,int(number**0.5)+1):
            if number% i== 0:
                raise valueerror('number is equal to zero so it is primenumber')
    except valueerror as error:
        logging.error(str(error))
    else:
        logging.info('number is a prime')
```

[12]:
```
try:
    with open('test.tx','w') as f:
        f.write('this is my data to write')
except FileNotFound as e:
    logging.error('i am  getting into  the poo bec i diid not find  a file{}'.
 ↪format(e))
finally :
    f.close()
```

[18]:
```
import logging
logging.basicConfig(filename='try.txt', level=logging.
 ↪INFO,format='%l(evelname)s: %(message)s')
```

```
def is_prime(number):
    try:
        if number<2:
            raise valueerror('number must be greater than or equal to 2')
        for i in range(2,int(number**0.5)+1):
            if number% i== 0:
                raise valueerror('number is equal to zero so it is primenumber')
    except valueerror as error:
        logging.error(str(error))
    else:
        logging.info('number is a prime')
```

[ ]:
```
#Q5
```

[ ]:
```
'custom exception are the exception we can customize the expection as we want␣
 ↪the expection to perform '
```

[ ]:
```
'we do custom expection as every expection can cannot be built the main␣
 ↪oriented expection are other all expection are customize able according to␣
 ↪the programmer '
```

[19]:
```
import logging

class CustomException(Exception):
    pass

def divide_number(a, b):
    logging.basicConfig(filename='log.txt',level=logging.INFO,␣
 ↪format='%(levelname)s: %(message)s')
    try:
        if b == 0:
            raise CustomException("Division by zero is not allowed")
        result = a / b
        return result
    except CustomException as e:
        logging.error(str(e))
```

[ ]:
```
Q6
```

[25]:
```
import logging
logging.basicConfig(filename='age.txt',level=logging.INFO,format='%(levelname)s:
 ↪%(message)s')
class vaildateage(Exception):
    def __init__(self,msg):
        self.msg= msg
def validaetage(age):
    if age<0:
```

```
        raise validateage('enter age is less than zero')
    elif age>200 :
        raise validateage('enter age is high')
    else:
        logging.info('age is valid {}'.formate(e))
    try:
        age=int(input('enter your age'))
        validaetage(age)
    except validateage as e:
        logging.info(e)
```

[ ]: