EXPERIMENT NO : 4C

Python programs to create a menu driven application For built-in exceptions in python.

NAME : AKASH RAMKRIT YADAV		***ID.NO: VU4F2122016***
BATCH : A	***BRANCH : IT***	***DIV : A***

Aim :- python programs to create a menu driven application which should cover all . the built-in exceptions in python.

THEORY:

OUTPUT:

Python 3.11.0a4 (main, Mar 13 2023, 10:57:32) [MSC v.1929 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
#AKASH YADAV ID.NO:VU4F2122016 EXP: 4C DATE:6/3/2023

menu driven program;

A menu driven program in python is a program that obtains a choice from a user by displaying the menu. Then, it will perform some operations and print the result according to the user's choice.

Error in Python:

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which change the normal flow of the program.

Different types of exceptions in python:

In Python, there are several built-in exceptions that can be raised when an error occurs during the execution of a program. Here are some of the most common types of exceptions in Python:

- **SyntaxError:** This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.

- **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
- **NameError**: This exception is raised when a variable or function name is not found in the current scope.

- **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.

- **KeyError**: This exception is raised when a key is not found in a dictionary.

- **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.

- **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.

- **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.

- **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.

- **ImportError**: This exception is raised when an import statement fails to find or load a module.

These are just a few examples of the many types of exceptions that can occur in Python. It's important to handle exceptions properly in your code using try-except blocks or other error-handling techniques, in order to gracefully handle errors and prevent the program from crashing.

Difference between Syntax Error and Exceptions

Syntax Error: As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

Exceptions: Exceptions are raised when the program is syntactically correct, but the code results in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.

Try and Except Statement – Catching Exceptions

Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

Catching Specific Exception

A try statement can have more than one except clause, to specify handlers for different exceptions. Please note that at most one handler will be executed. For example, we can add IndexError in the above code. The general syntax for adding specific exceptions are –

```
try:
    # statement(s)
except IndexError:
    # statement(s)
except ValueError:
    # statement(s)
```

Example: *Catching specific exceptions in the Python*

- *Python3*

```
# Program to handle multiple errors with one
# except statement
# Python 3

def fun(a):
    if a < 4:

        # throws ZeroDivisionError for a = 3
        b = a/(a-3)

    # throws NameError if a >= 4
    print("Value of b = ", b)

try:
    fun(3)
    fun(5)

# note that braces () are necessary here for
```

```
# multiple exceptions
except ZeroDivisionError:
    print("ZeroDivisionError Occurred and Handled")
except NameError:
    print("NameError Occurred and Handled")
```

Output

```
ZeroDivisionError Occurred and Handled
```

If you comment on the line fun(3), the output will be

```
NameError Occurred and Handled
```

The output above is so because as soon as python tries to access the value of b, NameError occurs.

Try with Else Clause

In Python, you can also use the else clause on the try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

Finally Keyword in Python

Python provides a keyword finally, which is always executed after the try and except blocks. The final block always executes after the normal termination of the try block or after the try block terminates due to some exception.

Syntax:

```
try:
    # Some Code....


except:
    # optional block
    # Handling of exception (if required)


else:
```

```
    # execute if no exception


finally:
    # Some code .....(always executed)
```

Raising Exception

The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).

- Python3

```
# Program to depict Raising Exception



try:

    raise NameError("Hi there")  # Raise Error

except NameError:

    print ("An exception")

    raise # To determine whether the exception was raised or not
```

The output of the above code will simply line printed as “An exception” but a Runtime error will also occur in the last due to the raise statement in the last line. So, the output on your command line will look like

```
Traceback (most recent call last):
  File "/home/d6ec14ca595b97bff8d8034bbf212a9f.py", line 5, in
<module>
    raise NameError("Hi there")  # Raise Error
NameError: Hi there
```

Advantages of Exception Handling:

- **Improved program reliability**: By handling exceptions properly, you can prevent your program from crashing or producing incorrect results due to unexpected errors or input.
- **Simplified error handling**: Exception handling allows you to separate error handling code from the main program logic, making it easier to read and maintain your code.
- **Cleaner code:** With exception handling, you can avoid using complex conditional statements to check for errors, leading to cleaner and more readable code.
- **Easier debugging**: When an exception is raised, the Python interpreter prints a traceback that shows the exact location where the exception occurred, making it easier to debug your code.

Disadvantages of Exception Handling:

- **Performance overhead:** Exception handling can be slower than using conditional statements to check for errors, as the interpreter has to perform additional work to catch and handle the exception.

- **Increased code complexity**: Exception handling can make your code more complex, especially if you have to handle multiple types of exceptions or implement complex error handling logic.

- **Possible security risks:** Improperly handled exceptions can potentially reveal sensitive information or create security vulnerabilities in your code, so it's important to handle exceptions carefully and avoid exposing too much information about your program.

-

Overall, the benefits of exception handling in Python outweigh the drawbacks, but it's important to use it judiciously and carefully in order to maintain code quality and program reliability.

Built-in Exceptions

The below example shows built-in exceptions that are usually raised in Python:

Exception Description

ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails

Exception	**Base class for all exceptions**
EOFError .	Raised when the input() method hits an "end of file" . . condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indentation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt ..	Raised when the user presses Ctrl+c, Ctrl+z or . . . Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedError .	Raised when an abstract method requires an inherited class to override the method
OSError .	Raised when a system related operation causes an error
OverflowError .	Raised when the result of a numeric calculation is . too large
ReferenceError	Raised when a weak reference object does not exist

RuntimeError .	Raised when an error occurs that do not belong to any specific exceptions
StopIteration .	Raised when the next() method of an iterator has no further values

SyntaxError **Raised when a syntax error occurs**

TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError .	Raised when there is a wrong value in a specified data type
ZeroDivisionError .	Raised when the second operator in a division is zero

CODE:

```
def exception1(a):
    try:
        list1 =["AKASH","YADAV","RAM"]
```

```
        print(list1[a])
    except LookupError:
        print("Array index out of bound error")
    else:
        print("Element that you were searching for is:",list1[a])

def exception2(x):
    try:
        a = x/0
        print(a)
    except ArithmeticError:
        print("This is an invalid operation.")
    else:
        print("Sucessfully performed the operation")
        print(a)

def exception3(p,q):
    assert q!=0, "Division by zero is not defined"
    print("Division is:",p/q)

def exception4():
    try:
        import module_that_does_not_exist
    except ImportError:
        print ("This module does not exist")

def exception5(m):
    mydata={1:"AKASH",2:"VU4F2122016",3:"Mumbai"}
    try:
        print(mydata[m])
    except KeyError:
        print("Key not found")

print("This is a menu-driven program for all built-in exceptions")

while True:
    print("Main Menu")
    print("1)LookUp Error")
    print("2)Arithmetic Error")
    print("3)Assertion Error")
```

```
print("4)Import Error")
print("5)Key Error")
print("6)Exit")
choice = int(input("Enter the choice of built-in exception from main menu:"))

if choice == 1:
    print("Function for LookUP Error/Index out of range")
    num = int(input("Enter the index of element you are searching for:"))
    exception1(num)

elif choice == 2:
    print("Function for Arithmetic Error")
    num = int(input("Enter any number:"))
    exception2(num)

elif choice == 3:
    print("Function for Assertion Error")
    print("This function performs division of two numbers taken by the user")
    num1 = int(input("Enter no. 1:"))
    num2 = int(input("Enter no. 2:"))
    exception3(num1,num2)

elif choice == 4:
    print("Function for Import Error")
    print("This function tries to import an unexisting module and throws a error")
    exception4()

elif choice == 5:
    print("Function for Key Error")
    num = int(input("Enter the key number:"))
    exception5(num)

elif choice == 6:
    break

else:
    print("Enter the correct choice!!")
```

OUTPUT:

```
This is a menu-driven program for all built-in exceptions
Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:1
Function for LookUP Error/Index out of range
Enter the index of element you are searching for:0
AKASH
Element that you were searching for is: AKASH
Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:1
Function for LookUP Error/Index out of range
Enter the index of element you are searching for:6
Array index out of bound error

Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:2
Fynction for Arithmetic Error
Enter any number:3
This is an invalid operation.

Main Menu
1)LookUp Error
2)Arithmetic Error
```

```
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:3
Function for Assertion Error
This function performs division of two numbers taken by the user
Enter no. 1:4
Enter no. 2:2
Division is: 2.0

Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:3
Function for Assertion Error
This function performs division of two numbers taken by the user
Enter no. 1:4
Enter no. 2:0
Traceback (most recent call last):
  File "c:\Users\Tanu\Downloads\Telegram Desktop\Python\exp1_python.py", line
339, in <module>
    exception3(num1,num2)
  File "c:\Users\Tanu\Downloads\Telegram Desktop\Python\exp1_python.py", line
297, in exception3
    assert q!=0, "Division by zero is not defined"
           ^^^^
AssertionError: Division by zero is not defined

Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
```

```
Enter the choice of built-in exception from main menu:4
Function for Import Error
This function tries to import an unexisting module and throws a error
This module does not exist

Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:5
Function for Key Error
Enter the key number:6
Key not found

Main Menu
1)LookUp Error
2)Arithmetic Error
3)Assertion Error
4)Import Error
5)Key Error
6)Exit
Enter the choice of built-in exception from main menu:6
```