**GAUTAM BUDDHA UNIVERSITY**

**PYTHON PROGRAMMING PRACTICAL FILE**

**LAB CODE**: **CS-385**

**PROGRAM: B.Tech CSE**

**SEMESTER: 5**

**SUBMITTED TO**: MS. JYOTI KAURAV

**SUBMITTED BY**: SIDDHARTH CHAUDHARY (215/UCF/029)

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | EXPERIMENT | DATE | SIGN |
| 1. | WRITE A PROGRAM TO IMPLEMENT OPERATORS |  |  |
| 2. | WRITE A PROGRAM TO PRINT PATTERNS |  |  |
| 3. | WRITE A PROGRAM TO IMPLEMENT STRING OPERATIONS |  |  |
| 4. | WRITE A PROGRAM TO IMPLEMENT DIFFERENT WAY TO PRINT LIST |  |  |
| 5. | WRITE A PROGRAM TO IMPLEMENT STACK AND QUEUE |  |  |
| 6. | WRITE A PROGRAM TO IMPLEMENT DIFFERENT OPERATIONS IN DICTIONARY |  |  |
| 7. | WRITE A PROGRAM TO IMPLEMENT LAMBDA FUNCTION, MAP FUNCTION AND FILTER FUNCTION |  |  |
| 8. | WRITE A PROGRAM TO IMPLEMENT MULTIPLE INHERITANCE |  |  |
| 9. | WRITE A PROGRAM TO READ AND WRITE FILES |  |  |
| 10. | WRITE A PROGRAM FOR DATA ANALYSING BY IMPORTING FILES |  |  |
| 11. | WRITE A PROGRAM TO IMPLEMENT EXCEPTION HANDLING |  |  |

EXPERIMENT 1

OBJECTIVE – PROGRAM TO IMPLEMENT OPERATORS

# Arithmetic operators

x = 10

y = 5

addition = x + y

subtraction = x - y

multiplication = x \* y

division = x / y

floor\_division = x // y

modulus = x % y

exponentiation = x \*\* y

print(f"Arithmetic Operators:")

print(f"Addition: {addition}")

print(f"Subtraction: {subtraction}")

print(f"Multiplication: {multiplication}")

print(f"Division: {division}")

print(f"Floor Division: {floor\_division}")

print(f"Modulus: {modulus}")

print(f"Exponentiation: {exponentiation}\n")

# Comparison operators

a = 10

b = 20

equal = a == b

not\_equal = a != b

greater\_than = a > b

less\_than = a < b

greater\_than\_or\_equal = a >= b

less\_than\_or\_equal = a <= b

print(f"Comparison Operators:")

print(f"Equal: {equal}")

print(f"Not Equal: {not\_equal}")

print(f"Greater Than: {greater\_than}")

print(f"Less Than: {less\_than}")

print(f"Greater Than or Equal: {greater\_than\_or\_equal}")

print(f"Less Than or Equal: {less\_than\_or\_equal}\n")

# Logical operators

p = True

q = False

logical\_and = p and q

logical\_or = p or q

logical\_not = not p

print(f"Logical Operators:")

print(f"Logical AND: {logical\_and}")

print(f"Logical OR: {logical\_or}")

print(f"Logical NOT: {logical\_not}\n")

# Assignment operators

z = 5

z += 3 # Equivalent to z = z + 3

print(f"Assignment Operator (+=): {z}\n")

# Membership operators

fruits = ['apple', 'banana', 'orange']

is\_in = 'banana' in fruits

is\_not\_in = 'grape' not in fruits

print(f"Membership Operators:")

print(f"'banana' is in fruits: {is\_in}")

print(f"'grape' is not in fruits: {is\_not\_in}")

OUTPUT:



EXPERIMENT 2

OBJECTIVE: WRITE A PROGRAM TO PRINT PATTERNS

def print\_square\_pattern(n):

for i in range(n):

for j in range(n):

print("\*", end=" ")

print()

def print\_triangle\_pattern(n):

for i in range(1, n + 1):

for j in range(i):

print("\*", end=" ")

print()

def print\_inverted\_triangle\_pattern(n):

for i in range(n, 0, -1):

for j in range(i):

print("\*", end=" ")

print()

def print\_pyramid\_pattern(n):

for i in range(1, n + 1):

print(" " \* (n - i) + "\* " \* i)

def main():

print("Square Pattern:")

print\_square\_pattern(5)

print("\nTriangle Pattern:")

print\_triangle\_pattern(5)

print("\nInverted Triangle Pattern:")

print\_inverted\_triangle\_pattern(5)

print("\nPyramid Pattern:")

print\_pyramid\_pattern(5)

if \_\_name\_\_ == "\_\_main\_\_":

main()





EXPERIMENT 3

OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT STRING OPERATIONS

def string\_operations(input\_string):

# Length of the string

length = len(input\_string)

print(f"Length of the string: {length}")

# Concatenation

new\_string = input\_string + " Concatenated"

print(f"Concatenated string: {new\_string}")

# String repetition

repeated\_string = input\_string \* 3

print(f"String repetition: {repeated\_string}")

# String slicing

substring = input\_string[2:6]

print(f"Sliced substring (index 2 to 5): {substring}")

# String uppercase and lowercase

uppercase\_string = input\_string.upper()

lowercase\_string = input\_string.lower()

print(f"Uppercase string: {uppercase\_string}")

print(f"Lowercase string: {lowercase\_string}")

# String replace

replaced\_string = input\_string.replace('Python', 'Java')

print(f"String replace: {replaced\_string}")

# String find

substring\_index = input\_string.find('is')

print(f"Index of 'is': {substring\_index}")

# String count

substring\_count = input\_string.count('is')

print(f"Count of 'is': {substring\_count}")

# String split

words = input\_string.split(' ')

print(f"String split into words: {words}")

# String strip

stripped\_string = " leading and trailing spaces "

stripped\_result = stripped\_string.strip()

print(f"Stripped result: '{stripped\_result}'")

if \_\_name\_\_ == "\_\_main\_\_":

input\_string = "Python is a powerful programming language."

print(f"Original String: {input\_string}\n")

string\_operations(input\_string)

OUTPUT:



EXPERIMENT 4

OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT DIFFERENT WAYS TO PRINT LIST.

def print\_list\_methods(my\_list):

# Method 1: Using a simple loop

print("Method 1: Using a simple loop")

for item in my\_list:

print(item, end=" ")

print("\n")

# Method 2: Using list comprehension

print("Method 2: Using list comprehension")

print(" ".join([str(item) for item in my\_list]))

print()

# Method 3: Using map and join

print("Method 3: Using map and join")

print(" ".join(map(str, my\_list)))

print()

# Method 4: Using \* unpacking

print("Method 4: Using \* unpacking")

print(\*my\_list)

print()

# Method 5: Using the pprint module

import pprint

print("Method 5: Using the pprint module")

pprint.pprint(my\_list)

print()

# Method 6: Using the print function with sep parameter

print("Method 6: Using the print function with sep parameter")

print(\*my\_list, sep=" ")

print()

if \_\_name\_\_ == "\_\_main\_\_":

my\_list = [1, 2, 3, 4, 5]

print(f"Original List: {my\_list}\n")

print\_list\_methods(my\_list)

OUTPUT:



EXPERIMENT 5

OBJECTIVE: WRITE A PROGRAM STACK AND QUEUE

class Stack:

def \_\_init\_\_(self):

self.items = []

def is\_empty(self):

return len(self.items) == 0

def push(self, item):

self.items.append(item)

def pop(self):

if not self.is\_empty():

return self.items.pop()

else:

print("Stack is empty")

def peek(self):

if not self.is\_empty():

return self.items[-1]

else:

print("Stack is empty")

def size(self):

return len(self.items)

class Queue:

def \_\_init\_\_(self):

self.items = []

def is\_empty(self):

return len(self.items) == 0

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.is\_empty():

return self.items.pop(0)

else:

print("Queue is empty")

def peek(self):

if not self.is\_empty():

return self.items[0]

else:

print("Queue is empty")

def size(self):

return len(self.items)

# Example usage:

stack = Stack()

stack.push(1)

stack.push(2)

stack.push(3)

print("Stack:", stack.items)

print("Pop:", stack.pop())

print("Peek:", stack.peek())

print("Size:", stack.size())

queue = Queue()

queue.enqueue('a')

queue.enqueue('b')

queue.enqueue('c')

print("\nQueue:", queue.items)

print("Dequeue:", queue.dequeue())

print("Peek:", queue.peek())

print("Size:", queue.size())

OUTPUT:



EXPERIMENT 6

OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT OPERATIONS IN DICTIONARY

def dictionary\_methods(my\_dict):

# Method 1: Accessing elements by key

print("Method 1: Accessing elements by key")

key = 'name'

value = my\_dict[key]

print(f"The value for key '{key}': {value}\n")

# Method 2: Adding a new key-value pair

print("Method 2: Adding a new key-value pair")

new\_key = 'age'

new\_value = 25

my\_dict[new\_key] = new\_value

print(f"Updated dictionary: {my\_dict}\n")

# Method 3: Removing a key-value pair using pop

print("Method 3: Removing a key-value pair using pop")

removed\_value = my\_dict.pop('age')

print(f"Removed key-value pair: {removed\_value}")

print(f"Updated dictionary: {my\_dict}\n")

# Method 4: Getting a value with a default

print("Method 4: Getting a value with a default")

default\_value = my\_dict.get('country', 'Not available')

print(f"Value for key 'country': {default\_value}\n")

# Method 5: Iterating through keys

print("Method 5: Iterating through keys")

for key in my\_dict.keys():

print(key, end=" ")

print("\n")

# Method 6: Iterating through values

print("Method 6: Iterating through values")

for value in my\_dict.values():

print(value, end=" ")

print("\n")

# Method 7: Iterating through key-value pairs

print("Method 7: Iterating through key-value pairs")

for key, value in my\_dict.items():

print(f"{key}: {value}")

print("\n")

# Method 8: Clearing the dictionary

print("Method 8: Clearing the dictionary")

my\_dict.clear()

print(f"Cleared dictionary: {my\_dict}\n")

if \_\_name\_\_ == "\_\_main\_\_":

my\_dict = {'name': 'John', 'city': 'New York', 'occupation': 'Engineer'}

print(f"Original Dictionary: {my\_dict}\n")

dictionary\_methods(my\_dict)

OUTPUT:



EXPERIMENT 7

OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT LAMBDA, MAP AND FILTER FUNCTION

# Example 1: Lambda function

square = lambda x: x \*\* 2

cube = lambda x: x \*\* 3

print("Example 1: Lambda function")

print("Square of 5:", square(5))

print("Cube of 3:", cube(3))

print()

# Example 2: Map function

numbers = [1, 2, 3, 4, 5]

squared\_numbers = list(map(lambda x: x \*\* 2, numbers))

cubed\_numbers = list(map(lambda x: x \*\* 3, numbers))

print("Example 2: Map function")

print("Original numbers:", numbers)

print("Squared numbers:", squared\_numbers)

print("Cubed numbers:", cubed\_numbers)

print()

# Example 3: Filter function

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

odd\_numbers = list(filter(lambda x: x % 2 != 0, numbers))

print("Example 3: Filter function")

print("Original numbers:", numbers)

print("Even numbers:", even\_numbers)

print("Odd numbers:", odd\_numbers)

OUTPUT:



EXPERIMENT 8

OBJECTIVE: WRITE A PROGRAM TO IMPLEMENT MULTIPLE INHERITANCE

# Parent class 1

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def make\_sound(self):

pass # Placeholder method

# Parent class 2

class Flyable:

def fly(self):

print(f"{self.name} is flying.")

# Parent class 3

class Swimmable:

def swim(self):

print(f"{self.name} is swimming.")

# Child class inheriting from Animal and Flyable

class Bird(Animal, Flyable):

def make\_sound(self):

print(f"{self.name} is chirping.")

# Child class inheriting from Animal and Swimmable

class Fish(Animal, Swimmable):

def make\_sound(self):

print(f"{self.name} is bubbling.")

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage

bird = Bird("Sparrow")

bird.make\_sound()

bird.fly()

print()

fish = Fish("Goldfish")

fish.make\_sound()

fish.swim()

OUTPUT:



EXPERIMENT 9

OBJECTIVE: WRITE A PROGRAM TO READ AND WRITE FILES

# Writing to a file

def write\_to\_file(file\_name, content):

with open(file\_name, 'w') as file:

file.write(content)

print(f"Content written to {file\_name}\n")

# Reading from a file

def read\_from\_file(file\_name):

try:

with open(file\_name, 'r') as file:

content = file.read()

print(f"Content read from {file\_name}:\n{content}")

except FileNotFoundError:

print(f"File {file\_name} not found.")

except Exception as e:

print(f"An error occurred: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

file\_name = "sample\_file.txt"

content\_to\_write = "Hello, this is a sample file content.\nPython is awesome!"

# Writing to a file

write\_to\_file(file\_name, content\_to\_write)

# Reading from a file

read\_from\_file(file\_name)

OUTPUT:



EXPERIMENT 10

OBJECTIVE: WRITE A PROGRAM FOR DATA ANALYSING BY IMPORTING FILES

import pandas as pd

def analyze\_data(file\_path):

# Read data from a CSV file into a Pandas DataFrame

try:

df = pd.read\_csv(file\_path)

# Display basic information about the DataFrame

print("Data Overview:")

print(df.info())

# Display descriptive statistics

print("\nDescriptive Statistics:")

print(df.describe())

# Display the first few rows of the DataFrame

print("\nFirst Few Rows:")

print(df.head())

# Perform additional data analysis as needed

except FileNotFoundError:

print(f"File {file\_path} not found.")

except Exception as e:

print(f"An error occurred: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Specify the path to your CSV file

csv\_file\_path = "sample\_data.csv"

EXPERIMENT 11

OBJECTIVE – WRITE A PROGRAM FOR EXCEPTION HANDLING

def divide\_numbers(a, b):

try:

result = a / b

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

result = None

except TypeError as e:

print(f"Error: {e}")

result = None

else:

print(f"Result of {a} / {b}: {result}")

finally:

print("This block always executes.")

if \_\_name\_\_ == "\_\_main\_\_":

# Example 1: Division by zero

divide\_numbers(5, 0)

# Example 2: Correct division

divide\_numbers(10, 2)

# Example 3: TypeError

divide\_numbers("10", 2)

# Analyze the data

analyze\_data(csv\_file\_path)

OUTPUT:

