## PRACTICE SHEET 2 SOLUTION

Q1) b) To create isolated environments for different projects

Q2) b) python -m venv myenv

Q3) a) <class 'dict'>

Q4) b) or c)

Q5) a) True

Explanation: x is not y → True, x == y → False, x < y → True   
True and False or True → False or True → True

Q6)

a = 256

b = 256

print(a is b)

True

Explanation : Small integers (-5 to 256) are cached in Python.

c = 257

d = 257

print(c is d)

False (may vary based on interpreter)

Q7)

import math

# Input sides of the triangle

a = float(input("Enter side a: "))

b = float(input("Enter side b: "))

c = float(input("Enter side c: "))

# Calculate angles using cosine rule

A = math.acos((b\*\*2 + c\*\*2 - a\*\*2) / (2 \* b \* c))

B = math.acos((a\*\*2 + c\*\*2 - b\*\*2) / (2 \* a \* c))

C = math.acos((a\*\*2 + b\*\*2 - c\*\*2) / (2 \* a \* b))

# Convert to degrees

A\_deg = math.degrees(A)

B\_deg = math.degrees(B)

C\_deg = math.degrees(C)

# Round to next integer

angle\_A = math.ceil(A\_deg)

angle\_B = math.ceil(B\_deg)

angle\_C = math.ceil(C\_deg)

# Print the angles

print("Angle A:", angle\_A)

print("Angle B:", angle\_B)

print("Angle C:", angle\_C)

Q8)

def getChessSquareColor(column, row):

# If the column and row is out of bounds, return a blank string

if column < 1 or column > 8 or row < 1 or row > 8:

return ''

# If the even/oddness of the column and row match, return 'white':

if column % 2 == row % 2:

return 'white'

# If they don't match, then return 'black':

else:

return 'black'

Q9)

(a) Print imaginary part out of 2 + 3j

z = 2 + 3j

print("Imaginary part:", z.imag)

(b) Obtain conjugate of 4 + 2j

z = 4 + 2j

print("Conjugate:", z.conjugate())

(c) Print decimal equivalent of binary '1100001110'

binary\_str = '1100001110'

print("Decimal equivalent:", int(binary\_str, 2))

(d) Convert a float value 4.33 into a numeric string

f = 4.33

print("Numeric string:", str(f))

(e) Obtain integer quotient and remainder while dividing 29 with 5

quotient, remainder = divmod(29, 5)

print("Quotient:", quotient, "Remainder:", remainder)

(f) Obtain hexadecimal equivalent of decimal 34567

print("Hexadecimal:", hex(34567))

(g) Round-off 45.6782 to second decimal place

print("Rounded value:", round(45.6782, 2))

(h) Obtain 4 from 3.556 (using round)

print("Rounded:", round(3.556))

(i) Obtain 17 from 16.7844 (using round)

print("Rounded:", round(16.7844))

(j) Obtain remainder on dividing 3.45 with 1.22

print("Remainder:", 3.45 % 1.22)

Q10 )

def ordinalSuffix(number):

# 11, 12, and 13 have the suffix th:

if number % 100 in (11, 12, 13):

return str(number) + 'th'

# Numbers that end with 1 have the suffix st:

if number % 10 == 1:

return str(number) + 'st'

# Numbers that end with 2 have the suffix nd:

if number % 10 == 2:

return str(number) + 'nd'

# Numbers that end with 3 have the suffix rd:

if number % 10 == 3:

return str(number) + 'rd'

# All other numbers end with th:

return str(number) + 'th'

Q11)

l = int(input())

r = int(input())

minn = 2 \* l

maxx = 2 \* r

n = maxx - minn + 1

print(n)

Q12)

text = input("Enter the original text: ")  
oldText = input("Enter the text to be replaced: ")  
newText = input("Enter the replacement text: ")

result = ""  
i = 0  
n = len(text)  
old\_len = len(oldText)

while i < n:  
# Check if current position matches oldText  
if text[i:i+old\_len] == oldText:  
# Add newText to result  
result += newText  
# Skip ahead by length of oldText  
i += old\_len  
else:  
# Add current character to result  
result += text[i]  
i += 1

print("Modified text:")  
print(result)

Q13)

totalSeconds = int(input("Enter total seconds: "))

if totalSeconds == 0:

print("0s")

else:

# Calculate days, hours, minutes, seconds

days = totalSeconds // (24 \* 3600)

remaining\_seconds = totalSeconds % (24 \* 3600)

hours = remaining\_seconds // 3600

remaining\_seconds %= 3600

minutes = remaining\_seconds // 60

seconds = remaining\_seconds % 60

# Build the output string

parts = []

if days > 0:

parts.append(f"{days}d")

if hours > 0:

parts.append(f"{hours}h")

if minutes > 0:

parts.append(f"{minutes}m")

if seconds > 0 or not parts: # Ensure at least '0s' if all zero

parts.append(f"{seconds}s")

print(' '.join(parts))

Q14)

# Define the circuit connections

resistor1 = "10"

parallel\_resistors = ["20", "30"]

# Build the parallel part (square brackets)

parallel\_part = "[" + " ".join(parallel\_resistors) + "]"

# Combine with series connection (curly braces)

circuit\_expression = "{" + resistor1 + " " + parallel\_part + "}"

# Print the final expression

print("Circuit Expression:", circuit\_expression)

Q15)

import re

expr = input("Enter a mathematical expression: ").strip()

# Check for invalid characters (only digits, +, -, \*, /, (, ), spaces allowed)

if not re.fullmatch(r'^[\d+\-\*/().\s]+$', expr):

print("Error: Invalid characters in expression")

else:

# Check for balanced brackets

stack = []

for char in expr:

if char == '(':

stack.append(char)

elif char == ')':

if not stack:

print("Error: Mismatched Brackets")

exit()

stack.pop()

if stack:

print("Error: Mismatched Brackets")

else:

# Check for operator errors (e.g., "5 + \* 3")

if re.search(r'[\+\-\\*/]\s\*[\+\-\\*/]', expr):

print("Error: Invalid Operator Usage")

else:

try:

result = eval(expr) # Evaluate using BODMAS

print(result)

except:

print("Error: Invalid Expression")