UNIT 3 Answer Key

1.What will be the output of the following programs?

```
def fun(x, y) :
    if x == 0 :
        return y
    else :
        return fun(x - 1, x * y)
print(fun(4, 2))
```

Output

48

2.Write a recursive function to obtain average of all numbers present in a given list.

```
def get_average(lst, n) :
        if n == 1 :
                return lst[ 0 ]
        else :
                return (get_average(lst, n - 1) * (n - 1) + lst[n - 1]) / n
                print(sum, n)
        return sum
numlst = [10, 20, 30, 40, 50, 60]
avg = get_average(numlst, len(numlst))
print(avg)
```

Output

35.0

3. A list contains some negative and some positive numbers. Write a recursive function that sanitizes the list by replacing all negative numbers with 0.

```
def replace(lst, i, n) :
        if i > n :
                return
        if lst[ i ] < 0 :
                lst[ i ] = 0
                i += 1
replace( lst, i, n)
numlst = [10, 20, -3, -4, 50, -4, 60, 70, -4]
replace(numlst, 0, len(numlst) - 1)
print(numlst)
```

Output

[10, 20, 0, 0, 50, 0, 60, 70, 0]

4.What will be the output of the following program

```
def fun(num) :
```

```
        if num > 100 :
                return num - 10
        return fun(fun(num + 11))
print(fun(75))
```

Output

91

5. Write a recursive program to check whether the given no is a power of two

```
def fun(num) :
        if num == 0 :
                print("False")
        if num == 1 :
                print("True")
        if num % 2 == 0 :
                fun(num / 2)
        else:
                return False
fun(32)
```

Output

True

6. Write a recursive Python function to calculate the factorial of a number.

```
def factorial(n):
   if n == 0 or n == 1:
      return 1
   else:
      return n * factorial(n-1)
```

7. Write a python program to implement a recursive function to calculate the nth Fibonacci number.

```
def fibonacci(n):
   if n <= 1:
      return n
   else:
      return fibonacci(n-1) + fibonacci(n-2)
```

8. Write a recursive Python function to find the greatest common divisor (GCD) of two numbers.

```
def gcd(a, b):
   if b == 0:
      return a
   else:
      return gcd(b, a % b)
```

9. Write a python program to implement a recursive function to reverse a string in Python.

```
def reverse_string(s):
   if not s:
      return s
   else:
      return reverse_string(s[1:]) + s[0]
```

10. Write a recursive Python function to calculate the sum of digits in a given number.

```
def sum_of_digits(n):
   if n < 10:
      return n
```

```
    else:
        return n % 10 + sum_of_digits(n // 10)
```

11. Write a python program to implement a recursive function to find the power of a number (x^n) in Python.

```
def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)
```

12.State whether the following statements are True or False:

a. If a recursive function uses three variables **a**, **b** and **c**, then the same set of variables are used during each recursive call.

True

c. Multiple copies of the recursive function are created in memory.

False

d. A recursive function must contain at least 1 **return** statement.

True

e. Every iteration done using a **while** or **for** loop can be replaced with recursion.

True

h. Infinite recursion can occur if the base case is not properly defined.

True

i. A recursive function is easy to write, understand and maintain as compared to a one that uses a loop.

False

13.Write a program that uses a generator to create a set of unique words from a line input through the keyboard.

```
def unique_words_generator(line):
    seen_words = set()
    for word in line.split():
        if word not in seen_words:
            seen_words.add(word)
            yield word

input_line = input("Enter a line of text: ")
unique_words = set(unique_words_generator(input_line))

print("\nUnique words in the input line:")
for word in unique_words:
        print(word)
```

Output

```
Enter a sentence: I did not do this. He did it or she did it
{'it', 'or', 'do', 'He', 'she', 'did', 'this.', 'I', 'not'}
```

14.Write a program that uses a generator to find out maximum marks obtained by a student and his name from tuples of multiple students.

```
def max_marks_generator(students):
    max_marks = 0
    max_student_name = ""
```

```
    for student in students:
        name, marks = student
        if marks > max_marks:
            max_marks = marks
            max_student_name = name

      yield max_student_name, max_marks

  # Sample data with tuples of students (name, marks)
  student_data = [("Anu", 70), ("Bijoy", 75), ("Chris", 95), ("Diwakar", 98)]
  # Use the max_marks_generator to find the student with the maximum marks
  result_generator = max_marks_generator(student_data)
  # Extract the result from the generator
  result = next(result_generator)
  # Display the result
  print(f"The student with the maximum marks is {result[0]} with {result[1]} marks.")
```
The student with the maximum marks is Charlie with 95 marks.

15. Implement a recursive Python function to check if a given word is a palindrome.

```
def is_palindrome(word):
    if len(word) <= 1:
        return True
    else:
        return word[0] == word[-1] and is_palindrome(word[1:-1])
```

16. Write a program that uses a generator that generates characters from a string in reverse order.

```
def reverse_char_generator(input_str):
    for char in reversed(input_str):
        yield char

input_string = input("Enter a string: ")

# Use the reverse_char_generator to generate characters in reverse order
reversed_characters = reverse_char_generator(input_string)

# Display the reversed characters
print("Characters in reverse order:")
for char in reversed_characters:
        print(char, end=" ")
```

Output

Enter a string: Sacchidanand'
['d', 'n', 'a', 'n', 'a', 'd', 'i', 'h', 'c', 'c', 'a', 'S']

17. What is the difference between the following statements?

```
sum([x**2 for x in range(20)])
sum(x**2 for x in range(20))
```

The first expression first generates a list and then obtains the sum of all elements in the list.

The second expression keeps a running sum of square of each number
generates as and when they get generated.
Both will yield same result, but the second one is more efficient as it
occupies less space.
18. Suppose there are two lists, each holding 5 strings. Write a closure program to generate a list that consists of strings that are concatenated by picking corresponding elements from the two lists.

```
def concat_strings_closure(list1, list2):
    def concatenate():
        result = []
        for str1, str2 in zip(list1, list2):
            result.append(str1 + str2)
        return result
    return concatenate

# Sample data with two lists of strings
list1 = ["apple", "banana", "cherry", "date", "fig"]
list2 = ["orange", "grape", "kiwi", "lemon", "mango"]

# Use the concat_strings_closure to create a generator function
concatenate_func = concat_strings_closure(list1, list2)

# Call the generator function to get the concatenated list
concatenated_list = concatenate_func()

# Display the concatenated list
print(concatenated_list)
```

19. Explain what a closure is in Python and provide an example of a closure.
A closure in Python is a function object that has access to variables in its lexical scope, even when the function is called outside that scope. It allows a function to remember and use the variables from the outer (enclosing) function.
Example:

```
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

closure_example = outer_function(10)
result = closure_example(5)  # This will result in 15
```

20.How does a closure differ from a regular function in Python?
A closure is a function object that has access to variables in its lexical scope, even when the function is called outside that scope. In contrast, a regular function in Python does not retain access to the variables from its calling scope once the function execution is complete.

21. Explain the concept of a free variable in the context of closures.
A free variable in a closure is a variable that is not bound within the closure itself but is accessed from its containing (enclosing) scope. The closure "closes over" this variable, allowing it to retain its value even after the enclosing function has finished execution.

22. How can closures be used to implement data hiding in Python? Provide an example.
Closures can be used to encapsulate data and provide a form of data hiding by restricting direct access to certain variables.
Example:

```
def counter():
    count = 0
    def increment():
        nonlocal count
        count += 1
        return count
    return increment

counter_instance = counter()
result = counter_instance()  # This will result in 1
```

23. Discuss a scenario where using closures in Python can be beneficial.
Closures are beneficial when you want to create functions with behavior that depends on an external context, and you want to encapsulate that behavior. For example, in event handling or callback functions, closures can be used to maintain the state between function calls.

24. Explain the term "lexical scoping" in the context of closures.
Lexical scoping refers to the way the scope of variables is determined by their position within the code. In the context of closures, it means that the variables accessed by the closure are based on the location of the variable definition in the source code.

25. Give an example of a closure that takes multiple arguments.

```
def multiplier(x):
    def inner_multiplier(y):
        return x * y
    return inner_multiplier

closure_example = multiplier(5)
result = closure_example(3)  # This will result in 15
```

26. How does the lifespan of a variable in a closure compare to the lifespan of a variable in a regular function?
Variables in a closure can outlive the lifespan of a regular function because the closure retains access to the variables from its enclosing scope even after the function has finished execution.

27. State whether the following statements are True or False:

A decorator adds some features to an existing function.
True
Once a decorator has been created, it can be applied to only one function within the program.
False
It is mandatory that the function being decorated should not receive any arguments.
False
It is mandatory that the function being decorated should not return any value.

False

28.Explain what a callback function is in Python and provide an example.
A callback function in Python is a function that is passed as an argument to another function and is executed after the completion of a specific event. It allows for asynchronous and event-driven programming.
Example:

```
def callback(message):
    print(f"Callback received: {message}")

def call(callback, data):
    result = data * 2
    # Callback
    callback(result)
call(callback,5) # This will print "Callback received: 10"
```

29. Illustrate an example of callback in the context of handling user input in a graphical user interface (GUI) application.

```
from tkinter import *
def on_button_click():
    print("Button clicked!")
root = Tk()
button = Button(root, text="Click me", command=on_button_click)
button.pack()
root.mainloop()
```

30. Explain the concept of a decorator in Python and give an example of a simple decorator function.
A decorator in Python is a function that takes another function and extends or modifies its behavior. It is commonly used to wrap or decorate functions. Here's an example:

```
def simple_decorator(func):
    def wrapper():
        print("Before function execution")
        func()
        print("After function execution")
    return wrapper

@simple_decorator
def say_hello():
    print("Hello, world!")

say_hello()
```

Output:

```
Before function execution
Hello, world!
After function execution
```

31. Discuss the concept of nested decorators in Python and provide an example.
Nested or chain of decorators involve applying multiple decorators to a single function. Each decorator is applied in the order they appear. Here's an example:

```
def decorator_one(func):
    def wrapper():
        print("Decorator One")
        func()
    return wrapper

def decorator_two(func):
    def wrapper():
        print("Decorator Two")
        func()
    return wrapper

@decorator_one
@decorator_two
def decorated_function():
    print("Decorated Function")

decorated_function()
```

Decorator One
Decorator Two
Decorated Function

32. Explain the concept of a generator in Python and provide an example of a simple generator function.
A generator in Python is a special type of iterator that allows you to iterate over a potentially large sequence of data without loading the entire sequence into memory. It uses the yield keyword to produce a series of values lazily.

```
def simple_generator():
    yield 1
    yield 2
    yield 3

gen = simple_generator()
for value in gen:
    print(value)
# This will print:
# 1
# 2
# 3
```

33.Discuss the advantages of using generators over lists in terms of memory efficiency.
Generators are memory-efficient because they produce values on-the-fly and do not store the entire sequence in memory. This is particularly beneficial when dealing with large datasets or infinite sequences.

34. Give an example of using a generator expression to generate a sequence of square numbers.

```
square_numbers = (x**2 for x in range(5))
for num in square_numbers:
    print(num)
# This will print:
# 0
# 1
# 4
# 9
# 16
```

35.Question: Explain the difference between a generator function and a regular function
In a generator, function uses the yield keyword to produce a sequence of values lazily, allowing for memory-efficient iteration. In contrast, a regular function returns a single value and terminates when the return statement is encountered.

36.Discuss the role of the next function in iterating over generator objects in Python.
The next function is used to retrieve the next value from a generator. It allows for controlled iteration and is useful when working with large or infinite sequences.

37. Question: Give an example of using a generator to implement the Fibonacci sequence in Python.

```
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fib_gen = fibonacci_generator()
for _ in range(5):
    print(next(fib_gen))
# This will print the first 5 numbers of the Fibonacci sequence.
```

38. State True/False
1 True or False? Modular design allows programs to be broken down into manageable size parts, in which each module (part) is a function with clearly specified functionality.
Answer:False
2 True or False? A module may be just the design of specific functionality, without any implementation.
Answer:True
3 True or False? A module, in terms of program design, may consist of a single function.
Answer:True
4 True of False? Modules are useful for the design, development, testing and maintenance of computer programs.
Answer:True
5 True or False? The specification of a module is referred to as the module's

interface.
Answer:True
6 True or False? A client should always be knowledgeable of the implementation of a module using, and not rely soley on the module's specification (interface).
Answer:False
7 True or False? A docstring is a specific feature of Python for providing the specification of program elements (e.g., functions and modules).
Answer:True
8 True or False? The __doc__ extension in Python is used in the creation of docstrings.
Answer:False
9 True or False? A module in Python is a file containing definitions and statements, which must contain a module declaration statement as the first line.
Answer:False
10 True or False? Modules in Python have their own namespace.
Answer:True
11 True or False? A name clash results when the same identifier is used in more than one function of a given program.
Answer:False
12 True or False? A fully qualified identifier is an identifier with the name of the module that it belongs to prepended to it.
Answer:True
13 True or False? The main module of a Python program is the one that is directly executed when a program is run.
Answer:True

39.For the following module,
# module driving_conversions
def milesPerHr(km_speed):
'''Returns miles per hour for kilometers per hour given in speed. '''
def milesPerGal(KilometersPerLiter):
'''Returns miles per gallon for provided kilometers per liter. '''
(a) Provide a main module that makes use of this module to prompt the user for the conversion desired,
and displays the converted result, using the import modulename form of import.
(b) Modify (a) above using the from modulename import identifier, identifier form of import.
(c) Modify (a) above using the from modulename import * form of import.
(d) Modify (a) above using the from modulename import identifier as new_identifier
form of import.
(e) Describe the namespaces for the main modules of (a), (b), (c) and (d) above.

ANSWERS:

(a)
```
import driving_conversions
response = int(input('(1) Convert kilometers per hour, ' + '(2) Convert liters per gallon: '))
while response not in (1, 2):
        response = int(input('(1) Convert kilometers per hour, ' + '(2) Convert liters per gallon:'))
if response == 1:
```

```
        kph = int(input('\nEnter speed (in kilometers per hour): '))
        print(kph, 'kilometers per hour equals',format(driving_conversions.milesPerHr(kph), '.2f'),'miles per
hour')
else:
        kpl = int(input('\nEnter fuel usage (in kilometers per liter): '))
        print(kpl, 'kilometers per liter equals',format(driving_conversions.milesPerGal(kpl), '.2f'),'miles per
gallon')
```

(b)

```
from driving_conversions import milesPerHr, milesPerGal
# ---- main
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
while response not in (1, 2):
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
if response == 1:
kph = int(input('\nEnter speed (in kilometers per hour): '))
print(kph, 'kilometers per hour equals',
format(milesPerHr(kph), '.2f'),
'miles per hour')
else:
kpl = int(input('\nEnter fuel usage (in kilometers per liter): '))
print(kpl, 'kilometers per liter equals',
format(milesPerGal(kpl), '.2f'),
'miles per gallon')
```

(c)

```
from driving_conversions import *
# ---- main
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
while response not in (1, 2):
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
if response == 1:
kph = int(input('\nEnter speed (in kilometers per hour): '))
print(kph, 'kilometers per hour equals',
format(milesPerHr(kph), '.2f'),
'miles per hour')
else:
kpl = int(input('\nEnter fuel usage (in kilometers per liter): '))
print(kpl, 'kilometers per liter equals',
format(milesPerGal(kpl), '.2f'),
'miles per gallon')
```

(d)

```
from driving_conversions import milesPerHr as MPH, milesPerGal as MPG
# ---- main
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
while response not in (1, 2):
response = int(input('(1) Convert kilometers per hour, ' + \
'(2) Convert liters per gallon: '))
if response == 1:
kph = int(input('\nEnter speed (in kilometers per hour): '))
print(kph, 'kilometers per hour equals',
```

format(MPH(kph), '.2f'),
'miles per hour')
else:
kpl = int(input('\nEnter fuel usage (in kilometers per liter): '))
print(kpl, 'kilometers per liter equals',
format(MPG(kpl), '.2f'),
'miles per gallon')
(d) The namespace for the main module in (a) above contains the following identifiers:
response, kph, kpl, driving_conversions, (built-in namespace)
The namespace for the main module in (b) above contains the following identifiers:
response, kph, kpl, milesPerHr, milesPerGal, (built-in namespace)
The namespace for the main module in (c) above contains the following identifiers:
response, kph, kpl, milesPerHr, milesPerGal, (built-in namespace)
The namespace for the main module in (c) above contains the following identifiers:
response, kph, kpl, MPH, MPG, (built-in namespace)

40. Fill in the Blanks
1._____ extension in Python is used in the creation of docstrings.
Answer:__doc__
2._____ is used to create special directories (folders) for Python modules to be placed.
3._____of a given module cannot be accessed by any importing module.
Answer: Privates identifiers (variables)
41.How does a module source code file become a module object?
Answer:
A module's source code file automatically becomes a module object when that module is imported. Technically, the module's source code is run during the import, one statement at a time, and all the names assigned in the process become attributes of the module object.
42. What is a package?
Packages are namespaces that contain multiple packages and modules themselves. They are simply directories.
import <modulename>
43.Write a Python script to display the current date and time.
import datetime
print("date and time", datetime.datetime.now())
44. What is the special file that each package in Python must contain?
Each package in Python must contain a special file called __init__.py. __init__.py can be an empty file but it is often used to perform setup needed for the package(import things, load things into path, etc).
Example :
package/
__init__.py
file.py
file2.py
file3.py

subpackage/
__init__.py
submodule1.py
submodule2.py