الاسم : احمد السيد عبده على فرج | سكشن 1

Chapter 3

Multiple Choice Questions

9 Which of the following is a characteristic of a pure function?
a) Depends on global variables
b) Produces side effects
c) Always returns the same output for the same input
d) Modifies input arguments

10 Which functional programming concept ensures that once a variable is assigned, it cannot be changed?
a) Recursion
b) Immutability
c) Closures
d) Memoization

11 Which built-in function applies a function to all elements of an iterable and returns an iterator?
a) filter()
b) reduce()
c) map()
d) zip()

12 Which module provides the reducefunction in Python?
a) operator
b) itertools
c) functools
d) collections

13 The filter()function in Python returns:
a) A list of all elements
b) An iterator containing elements that satisfy the condition
c) A tuple of matching elements
d) None
ADVANCED PROGRAMMING WITH PYTHON
True/False Questions
14 Functional programming in Python encourages immutability and pure functions.
Answer = T

15 The map() function modifies the original iterable in place.
Answer = F

16 Closures allow inner functions to access variables from their enclosing function even after the outer function has finished execution.
Answer = T

17 The reduce() function is a built-in function in Python and does not require an import.
Answer = F

18 itertools provide memory-efficient tools for working with iterators, including infinite sequences.
Answer = T

```
#----------------------
#----  problem 1  -----
#----------------------

# def remove_vowels(str):
#     vowels = "aeiouAEIOU"
#     return "".join([char for char in str if char not
in vowels])

##########################################
#----------------------
#----  problem 2  -----
#----------------------

# nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# odd_nums = filter(lambda x: x % 2 != 0, nums)
# squared_odds = list(map(lambda x: x**2, odd_nums))

##########################################
#----------------------
#----  problem 3  -----
#----------------------

# import functools
# import time
```

```
# @functools.lru_cache(maxsize=None)
# def fibonacci(n):
#     if n < 2:
#         return n
#     return fibonacci(n-1) + fibonacci(n-2)
# start_time = time.time()
# print(f"Fibonacci(35) = {fibonacci(35)}")
# end_time = time.time()
# print(f"Time taken: {end_time - start_time:.5f}
seconds")

##############################################
#----------------------
#----  problem 4  -----
#----------------------

# def make_adder(n):
#     def adder(x):
#         return x + n
#     return adder

##############################################
#----------------------
#----  problem 5  -----
#----------------------

# def apply_twice(func, value):
#     return func(func(value))
# print(apply_twice(lambda x: x + 1, 5))

##############################################
#----------------------
#----  problem 6  -----
#----------------------

# def etl_pipeline(texts):
#     stopwords = {"the", "a", "is", "in", "at", "of",
"on", "and", "to", "for"}
#     all_Words = [word.lower() for text in texts for
word in text.split()]
#     filtered_Words = filter(lambda word: word and word
not in stopwords, all_Words)

#     word_Frequencies  = {}
#     for word in filtered_Words:
#         word_Frequencies[word] =
word_Frequencies.get(word, 0) + 1
```

```
#     return word_Frequencies

##########################################
#----------------------
#----  problem 7  -----
#----------------------

# def my_reduce(func, iterable, initializer=None):
#     iterator = iter(iterable)
#     if initializer is None:
#         try:
#             value = next(iterator)
#         except StopIteration:
#             raise TypeError("Empty sequence with no
initial value")
#     else:
#         value = initializer

#     for element in iterator:
#         value = func(value, element)

#     return value

##########################################
#----------------------
#----  problem 8  -----
#----------------------

# def log_call(func):
#     def nestedFunc(*args, **kwargs):
#         print(f"--- Calling function: {func.__name__}
---")
#         result = func(*args, **kwargs)
#         print(f"--- Finished function: {func.__name__}
---")
#         return result
#     return nestedFunc
```