MANISA CELAL BAYAR UNIVERSITY – DEPARTMENT OF COMPUTER ENGINEERING
PROBLEM SET FOR PARALLEL PROGRAMMING

WEEK 02: DATA STRUCTURES IN PYTHON

1. What is the correct writing of the programming language that we used in this course?
( ) Phyton
( ) Pyhton
( ) Pthyon
( ) Python

2. What is the output of the code below?
```
my_name = "Bora Canbula"
print(my_name[2::-1])
```
( ) alu
( ) ula
( ) roB
( ) Bor

3. Which one is not a valid variable name?
( ) for_
( ) Manisa_Celal_Bayar_University
( ) IF
( ) not

4. What is the output of the code below?
```
for i in range(1, 5):
  print(f"{i:2d}{(i/2):4.2f}", end='')
```
( ) 010.50021.00031.50042.00
( ) 10.50 21.00 31.50 42.00
( ) 1 0.5 2 1.0 3 1.5 4 2.0
( ) 100.5 201.0 301.5 402.0

5. Which one is the correct way to print Bora's age?
```
profs = [
  {"name": "Yener", "age": 25},
  {"name": "Bora", "age": 37},
  {"name": "Ali", "age": 42}
]
```
( ) profs[“Bora”][“age”]
( ) profs[1][1]
( ) profs[1][“age”]
( ) profs.age[name=“Bora”]

6. What is the output of the code below?
```
x = set([int(i/2) for i in range(8)])
print(x)
```
( ) {0, 1, 2, 3, 4, 5, 6, 7}
( ) {0, 1, 2, 3}
( ) {0, 0, 1, 1, 2, 2, 3, 3}
( ) {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4}

7. What is the output of the code below?
```
x = set(i for i in range(0, 4, 2))
y = set(i for i in range(1, 5, 2))
print(x^y)
```
( ) {0, 1, 2, 3}
( ) {}
( ) {0, 8}
( ) SyntaxError: invalid syntax

8. Which of the following sequences is immutable?
( ) List
( ) Set
( ) Dictionary
( ) String

9. What is the output of the code below?
```
print(int(2_999_999.999))
```
( ) 2
( ) 3000000
( ) ValueError: invalid literal
( ) 2999999

10. What is the output of the code below?
```
x = (1, 5, 1)
print(x, type(x))
```
( ) [1, 2, 3, 4] <class ‘list’>
( ) (1, 5, 1) <class ‘range’>
( ) (1, 5, 1) <class ‘tuple’>
( ) (1, 2, 3, 4) <class ‘set’>

WEEK 03: FUNCTIONS AND DECORATORS

1. Does a Python function always return a value?

( ) True
( ) False

2. Which of the following is the valid start to define a function in Python?

( ) define func():
( ) function func() {
( ) void func():
( ) def func():

3. What does return from call mltpl(2,3)?

```
def mltpl(a, b=1):
    return a*b
```

Your Answer:

4. How can you use the following function to print exactly 'ParallelProgramming'?

```
def a(x):
    def b(y):
        print(y, end='')
    print(x, end='')
```

( ) a('Parallel Programming')
( ) a('Parallel');b('Programming')
( ) a('Parallel');a('Programming')
( ) a.b('ParallelProgramming')

5. How can you change 'BC' with your own initials in the following function?

```
def speak(s):
    if not speak.who:
        speak.who = 'BC'
    print(f"{speak.who} says {s}")
```

Your Answer:

6. There is a module called 'logging' to employ logging facility in Python.

```
import logging
logging.info('Just a normal message')
logging.warning('Not fatal but still noted')
logging.error('There is something wrong')
```

You are expected to implement logging feature to an existing code which uses the function below.

```
def my_ugly_debug(s, level=0):
    pre_text = [
        "INFO",
        "WARNING",
        "ERROR"
    ]
    print(f"{pre_text[level]}: {s}")
```

You are not allowed to make changes in my_ugly_debug, so find another way.

WEEK 04: TESTS, LOGGING, AND COROUTINES

1. Write the sum_of_digits function which satisfies the tests given below.

```python
def sum_of_digits(n: int) -> int:
    pass


if __name__ == "__main__":
    # tests for integer values
    tests = [[1, 1], [23, 5], [1001, 2], [5623, 16]]
    for x in tests:
        if not sum_of_digits(x[0]) == x[1]:
            str = f"Value test is failed for {x[0]}"
            exit(str)
    # tests for non-integer values
    tests = [1.5, 1 + 2j, "a", True]
    for x in tests:
        if not sum_of_digits(x) == TypeError:
            str = f"Type test is failed for {x}"
            exit(str)
    print("Tests are completed.")
```

2. Rewrite the test part of the code given in question **1** by using logging module.

3. The url www.canbula.com/prime/{n} returns a dictionary including the prime numbers below an integer n.
Example:
Request: https://www.canbula.com/prime/5
Response: {"n":"5","primes":[2,3]}

We want to test this service but the problem is response times are really long. Therefore you are requested to:

- Write tests for almost all scenerios
- Your tests should be running asynchronously so we don't have to wait for every test sequentially
- If you still have some extra time, develop your own project with Flask or FastAPI, which satisfies your tests.

WEEK 05: SCHEDULING WITH ASYNCIO

1. How can you classify a problem as CPU-bound or IO-bound?

2. When do we use context managers?

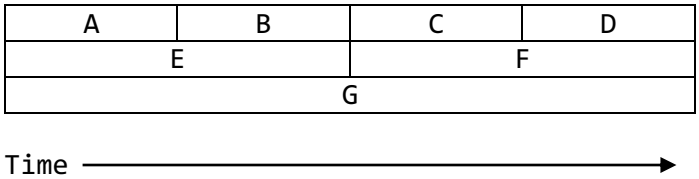
3. Code given below raises RuntimeWarning, correct the code to call the coroutine without an error.

```
import asyncio

async def an_async_func(delay: int) -> None:
    await asyncio.sleep(delay)
    print(f"U called me with "
          f"{delay} seconds delay.")
    return None

an_async_func(3)
```

4. We have coroutines A, B, C, D, E, F, G on a time scale given as:



Please write a main() coroutine which schedules the coroutines given as in the time scale.

WEEK 06: THREADS

1. Create two threads by using a target function and using a sub-class of Thread class from threading module. Emphasize the difference.

2. What is the difference between a non-daemon thread and a daemon thread?

3. Write a class which creates a thread from a coroutine.

4. Create a daemon thread which is continuously checking if a local file is changed by another program.

WEEK 07: GLOBAL INTERPRETER LOCK, RACE CONDITION, DEADLOCK

1. If Python is single-threaded, why do we create multiple threads and how does Python make us feel like it is running them simultaneously?

3. Which problem do we solve by using the just-in-time compiler from Numba?

2. Save my money from a possible race condition in the following code.

```
import threading

class Wallet:
    def __init__(self, money:int = 0):
        self.money = money

    def spend(self, amount):
        self.money -= amount

    def save(self, amount):
        self.money += amount

if __name__ == "__main__":
    wallet = Wallet()
    savers = [
        threading.Thread(
            target=wallet.save, args=(10,)
        )
        for _ in range(100000)
    ]
    spenders = [
        threading.Thread(
            target=wallet.spend, args=(10,)
        )
        for _ in range(100000)
    ]
    for saver, spender in zip(savers, spenders):
        saver.start()
        spender.start()
    for saver, spender in zip(savers, spenders):
        saver.join()
        spender.join()
    print(f"The total money in wallet: {wallet.money}")
```

4. Prevent these threads to be stuck by implementing a timeout feature.

```
import threading


class Resources(threading.Thread):
    def __init__(self, lock1, lock2):
        super().__init__()
        self.lock1 = lock1
        self.lock2 = lock2

    def run(self):
        with self.lock1:
            print(f"Thread {self.name} acquired lock1")
            with self.lock2:
                print(f"Thread {self.name} acquired lock2")


if __name__ == "__main__":
    lock1 = threading.Lock()
    lock2 = threading.Lock()
    t1 = Resources(lock1, lock2)
    t2 = Resources(lock2, lock1)
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```