MAY 29 Python ASSESSMENT

FUNCTIONS (Exercises 1–3)

1. Prime Number Checker

```
def is_prime(n):
    if n< 2:
        return False
    for i in range(2,n):
        if n%i== 0:
            return False
    return True
for num in range(1, 101):
    if is_prime(num):
        print(num, end=", ")
```

2. Temperature Converter

```
def convert_temp(value, unit):
    if unit == 'C':
        return (value*9/5)+ 32
    elif unit == 'F':
        return (value-32)* 5/9

print(convert_temp(10, 'C'))
print(convert_temp(32, 'F'))
```

3. Recursive Factorial Function

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

print(factorial(3))
```

4. Class: Rectangle

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

    def is_square(self):
        return self.length == self.width

rect = Rectangle(5, 5)
print("Area:", rect.area())
print("Perimeter:", rect.perimeter())
print("square", rect.is_square())
```

5. Class: BankAccount

```
class BankAccount:
    def __init__(self, name, balance=0):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        return f"Deposited {amount}. New balance: {self.balance}"

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        self.balance -= amount
        return f"Withdrew {amount}. Remaining balance: {self.balance}"

    def get_balance(self):
```

```
        return f"Current balance: {self.balance}"

account = BankAccount("Anderson Silva", 3500)
print(account.deposit(500))
print(account.withdraw(200))
print(account.withdraw(5000))
print(account.get_balance())
```

6. Class: Book

```
class Book:
    def __init__(self, title, author, price, in_stock):
        self.title = title
        self.author = author
        self.price = price
        self.in_stock = in_stock

    def sell(self, quantity):
        if quantity > self.in_stock:
            raise ValueError("Not enough stock")
        self.in_stock -= quantity
        return f"Sold {quantity} copies of '{self.title}'"

book=Book("Mongo DB", "Fathima", 50, 20)
try:
    print(book.sell(3))
    print(book.sell(23))
except ValueError as e:
    print("Error:", e)
```

7. Student Grade System

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def average(self):
        return sum(self.marks) / len(self.marks)
```

```
    def grade(self):
        avg= self.average()
        if avg>= 90: return 'A'
        if avg>= 80: return 'B'
        if avg>= 70: return 'C'
        return 'F'

student = Student("Kristen", [99,84,72,89,66])
print(f"{student.name}'s average: {student.average()}")
print(f"{student.name}'s grade: {student.grade()}")
```

INHERITANCE (Exercises 8–10)

8. Person → Employee

```
class Person:
    def __init__(self, name, age):
        self.name= name
        self.age= age

class Employee(Person):
    def __init__(self, name, age, emp_id, salary):
        super().__init__(name, age)
        self.emp_id= emp_id
        self.salary= salary

    def display_info(self):
        return (f"Name: {self.name}, Age: {self.age}, ID: {self.emp_id}, Salary: {self.salary}")

emp= Employee("Zahira",35,"101",500000)
print(emp.display_info())
```

9. Vehicle → Car, Bike

```
class Vehicle:
    def __init__(self, name, wheels):
        self.name = name
        self.wheels = wheels

    def description(self):
        return f"{self.name} with {self.wheels} wheels"
```

```
class Car(Vehicle):
    def __init__(self, name, wheels, fuel_type):
        super().__init__(name, wheels)
        self.fuel_type= fuel_type

    def description(self):
        return f"{super().description()}, Fuel: {self.fuel_type}"

class Bike(Vehicle):
    def __init__(self, name, wheels, is_geared):
        super().__init__(name, wheels)
        self.is_geared= is_geared

    def description(self):
        gear_status = "Geared" if self.is_geared else "Non-geared"
        return f"{super().description()}, Type:{gear_status}"

car = Car("Ford Taurus", 4, "Petrol")
bike = Bike("PCJ 600", 2, True)
print(car.description())
print(bike.description())
```

10. Polymorphism with Animals

```
class Animal:
    def speak(self):
        return "An animal sound"

class Dog(Animal):
    def speak(self):
        return "Woof"

class Cat(Animal):
    def speak(self):
        return "Meow"

class Cow(Animal):
```

```
    def speak(self):
        return "Mooooooo"

animals = [Dog(), Cat(), Cow()]
for animal in animals:
    print(animal.speak())
```