Answer Key

Lab 01: Terminal Commands & OOP Fundamentals

Comp 111 — Spring 2026

Only the student-written deliverable is shown. Accept reasonable variations.

Contents

1 Navigating the File System

1 ★E— `cd`

Prints current directory, e.g. `C:\Users\YourName`.

2 ★E— `dir`

Lists files/folders. Should see at least `Desktop`, `Documents`, `Downloads`.

3 ★E— `cls`

Screen clears. Prompt remains.

4 ★★M— Navigate to Desktop

```
cd Desktop
```

5 ★★M— Go back one folder

```
cd ..
```

6 ★★M— Absolute path

```
cd C:\Users\YourName\Desktop
```

7 ★★M— Documents then Desktop

```
cd ..\Documents
cd ..\Desktop
```

8 ★★★H— Relative from C:\

```
cd Users
cd YourName
cd Desktop
```

Also accept: `cd Users\YourName\Desktop`

2 Creating and Managing Folders

1 ★E

```
mkdir MyFirstFolder
```

2 ★E

`dir` — `MyFirstFolder` appears with `<DIR>`.

3 ★E

```
cd MyFirstFolder
```

4 ★★M

```
mkdir projects
cd projects
mkdir comp111
```

5 ★★M

```
cd ..\..\..
```

6 ★★M

```
mkdir lab01\src\animals
```

7 ★★★H— zoo_project structure

```
mkdir zoo_project
mkdir zoo_project\animals
mkdir zoo_project\utils
mkdir zoo_project\tests
```

3 Creating and Viewing Files

1 ★E

```
echo Zoo Management System > README.txt
type README.txt
```

2 ★E

```
echo By: YourName >> README.txt
```

3 ★★M

```
echo print("Welcome to the Zoo!") > main.py
```

4 ★★M

```
cd animals
echo print("Hello, I am an animal!") > hello_animal.py
```

5 ★★★H— info.txt with 3 lines

```
echo YourName > info.txt
echo 2024-FCCU-0001 >> info.txt
echo Comp 111 >> info.txt
```

4 Running Python from the Terminal

1–2 ★E

Guided. Student types `python`, experiments, `exit()`, then `python main.py`.

3 ★★M— adder.py

```
echo print(10 + 25) > adder.py
python adder.py
```

4 ★★M— greet.py

```python
name = input("What is your name? ")
print("Welcome to the Zoo, " + name + "!")
```

5 ★★★H— calculator.py

```python
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
print("Sum:", a + b)
print("Difference:", a - b)
print("Product:", a * b)
```

5 Classes and Objects

1 ★E— Point class

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

p = Point(3, 4)
print(p.x, p.y)
```

2 ★E— display method

```python
def display(self):
    print(f"({self.x}, {self.y})")
```

3 ★E— two points

```
p1 = Point(3, 4)
p2 = Point(7, 1)
p1.display()
p2.display()
```

4 ★★M— gpa methods

```
# Add to Student class:
self.gpa = 0.0          # in __init__

def set_gpa(self, gpa):
    self.gpa = gpa

def get_gpa(self):
    return self.gpa
```

5 ★★M— dean's list

```
def is_on_dean_list(self):
    return self.gpa >= 3.5
```

6 ★★M— Book class

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

    def discounted_price(self, percent):
        return self.price * (1 - percent / 100)
```

7 ★★M— Rectangle class

```
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)
```

8 ★★★H— BankAccount

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

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds!")
        else:
            self.balance -= amount

    def __str__(self):
        return f"Owner: {self.owner}, Balance: {self.balance}"
```

6 Data Abstraction and Encapsulation

1 ★E— use getters

```
p1 = Point(1, 2)
p2 = Point(4, 6)
print(p1.get_x(), p1.get_y())
print(p2.get_x(), p2.get_y())
```

2 ★E— distance function

```
def distance(p1, p2):
    dx = p1.get_x() - p2.get_x()
    dy = p1.get_y() - p2.get_y()
    return (dx**2 + dy**2) ** 0.5
```

3 ★★M— fix violation

Violation: `p.x` and `p.y` accessed directly.

```
# Fixed:
result = (p.get_x() ** 2 + p.get_y() ** 2) ** 0.5
```

4 ★★M— Rational class

```
import math

class Rational:
    def __init__(self, numer, denom):
        g = math.gcd(numer, denom)
        self.numer = numer // g
        self.denom = denom // g

    def get_numer(self):
        return self.numer
    def get_denom(self):
        return self.denom

    def __str__(self):
        return f"{self.numer}/{self.denom}"
```

5 ★★M— Rational add

```
def add(self, other):
    new_n = (self.get_numer() * other.get_denom() +
             other.get_numer() * self.get_denom())
    new_d = self.get_denom() * other.get_denom()
    return Rational(new_n, new_d)
```

6 ★★★H— Circle with composition

```
class Circle:
    def __init__(self, center, radius):
        self.center = center    # Point object
        self.radius = radius

    def contains_point(self, point):
        return distance(self.center, point) <= self.radius

    def __str__(self):
        return f"Circle(center={self.center}, radius={self.radius
            })"
```

Must use `distance()` — no direct `.x` / `.y` access.

7 Inheritance

1 ★E— Dog subclass

```
class Dog(Animal):
    def speak(self):
        print("woof")
```

2 ★E— inherited methods

```
d = Dog(3)
d.set_name("Buddy")
print(d.get_name())    # Buddy
print(d.get_age())     # 3
```

3 ★E— plain Animal

Prints . . . (the base class default).

4 ★★M— Person class

```
class Person(Animal):
    def __init__(self, name, age):
        Animal.__init__(self, age)
        self.set_name(name)
        self.friends = []

    def add_friend(self, friend_name):
        if friend_name not in self.friends:
            self.friends.append(friend_name)

    def speak(self):
        print("hello")
```

5 ★★M— Student class

```
class Student(Person):
    def __init__(self, name, age, major=None):
        Person.__init__(self, name, age)
        self.major = major

    def change_major(self, major):
        self.major = major

    def __str__(self):
        return f"student:{self.name}-{self.age}-{self.major}"
```

6 ★★M— Employee / Manager

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def calculate_pay(self):
        return self.salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        Employee.__init__(self, name, salary)
        self.bonus = bonus
    def calculate_pay(self):
        return self.salary + self.bonus
```

7 ★★M— parent can’t call child method

```
a = Animal(5)
# a.purr()
# AttributeError: 'Animal' object has no attribute 'purr'
```

8 ★★★H— Vehicle hierarchy + class variable

```
class Vehicle:
    vehicle_count = 0
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        Vehicle.vehicle_count += 1

class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        Vehicle.__init__(self, make, model, year)
        self.num_doors = num_doors
    def __str__(self):
        return f"{self.year} {self.make} {self.model} ({self.
            num_doors}-door)"

class Truck(Vehicle):
    def __init__(self, make, model, year, payload):
        Vehicle.__init__(self, make, model, year)
        self.payload = payload
    def __str__(self):
        return f"{self.year} {self.make} {self.model} (payload: {
            self.payload}kg)"
```

After 3 Cars + 2 Trucks: `Vehicle.vehicle_count == 5`.

8 Polymorphism

1 ★E— loop speak

```
for animal in animals:
    animal.speak()
```

Output: `meow`, `woof`, `meow`.

2 ★E— make_all_speak

```
def make_all_speak(animals):
    for animal in animals:
        animal.speak()
```

3 ★★M— Shape hierarchy + total_area

```
class Shape:
    def area(self):
        return 0

class Rectangle(Shape):
    def __init__(self, w, h):
        self.w = w
        self.h = h
    def area(self):
        return self.w * self.h

class Circle(Shape):
    def __init__(self, r):
        self.r = r
    def area(self):
        return 3.14159 * self.r ** 2

class Triangle(Shape):
    def __init__(self, b, h):
        self.b = b
        self.h = h
    def area(self):
        return 0.5 * self.b * self.h

def total_area(shapes):
    return sum(s.area() for s in shapes)
```

4 ★★M— print_payroll

```
# (Salesperson also needed)
class Salesperson(Employee):
    def __init__(self, name, salary, commission):
        Employee.__init__(self, name, salary)
        self.commission = commission
    def calculate_pay(self):
        return self.salary + self.commission

def print_payroll(employees):
    total = 0
    for emp in employees:
        pay = emp.calculate_pay()
        print(f"{emp.name}: ${pay}")
        total += pay
    print(f"Total: ${total}")
```

5 ★★M— duck typing

```
class Dog:
    def speak(self): print("woof")
class Robot:
    def speak(self): print("beep boop")
class Person:
    def speak(self): print("hello")

def make_noise(thing):
    thing.speak()
```

No inheritance between the three classes.

6 ★★★H— ABC Drawable + ASCII art

```
from abc import ABC, abstractmethod
```

```python
class Drawable(ABC):
    @abstractmethod
    def draw(self):
        pass

class Square(Drawable):
    def __init__(self, size):
        self.size = size
    def draw(self):
        for _ in range(self.size):
            print("* " * self.size)

class RightTriangle(Drawable):
    def __init__(self, h):
        self.h = h
    def draw(self):
        for i in range(1, self.h + 1):
            print("* " * i)

class Line(Drawable):
    def __init__(self, length):
        self.length = length
    def draw(self):
        print("* " * self.length)

def draw_all(shapes):
    for s in shapes:
        s.draw()
        print()
```

`Drawable()` raises `TypeError`. Accept any reasonable ASCII art.

9 Zoo Management System

cat.py / dog.py / parrot.py — student writes these

```python
# Each subclass file follows this pattern:
from animal import Animal

class Cat(Animal):           # (or Dog, Parrot)
    def speak(self):
        print("meow")        # (or "woof", "squawk!")
    def info(self):
        return f"Cat: {self.name} (age {self.age})"
```

zoo.py — student writes this

```python
import sys
sys.path.append("animals")
sys.path.append("utils")

from cat import Cat
from dog import Dog
from parrot import Parrot
from helpers import morning_routine

animals = [
    Cat("Simba", 3),  Dog("Buddy", 5),  Parrot("Polly", 10),
    Cat("Whiskers", 1),  Dog("Rex", 4),
]
morning_routine(animals)
```

Accept all files in one folder, different animal names, extra animal types.