

Name: Kelly Chinhamora

RegNo : H250394F

Department : Information Technology

Module : Principales of programming Languages

1. Compare procedural programming and functional programming, giving one code

example for each paradigm. (5 Marks)

**ANSWER**

|  |  |
| --- | --- |
| **Procedural programming**  **Defination**  It is a paradigm that focuses on how to perform tasks breaking them into small easy sequences of steps that can operate on shared data | **Functional programming**  **Definition**  It is a paradigm where computation is treated as the evaluation of mathematical functions |
| **Focus**  It includes a sequence of steps and procedures | **Focus**  It includes mathematical functions and expressions |
| **State**  It changes over time so it is mutable | **State**  It has no effects so it is not mutable |
| **Control structure**  Loops ,conditions | **Control structure**  Recursion , functions that are of high order |
| **Example**  def greet (name):      print("HELLO", name)  greet("KELLY")  #procedural is greet because it performs a specific action | **Example**  number =[1,2,3,4]  square=list(map(lambda  x:x/2,number)) #mapping every number in the list to the function  print(square) |

2. Write a recursive function to calculate the factorial of a number n. (4 Marks)

**Answer**

# A recursive function  is a programing technique where a function  calls itself to solve smaller instances of problem

def factorial(n):

    if n==0: # am using a base case of n ==0

        return 1

    else:

        return n \* factorial(n - 1)

print(factorial(6))

3. Explain the purpose of importing modules in a program and write a short code

snippet that imports the random module to generate a random number between 1 and

**Answer**

Importing modules allows user and makes it easy for them to use pre written codes in their program

import random

print (random.randint(1,10))

4. Define semantics and show code where syntax is correct but semantics are wrong. (4

Marks)

**ANSWER**

Semantics refers to the meaning of the program

Y= 20

X="KELLY"

print(X+Y)

#syntax is correct but the semantics are wrong because an integer can’t be added  with a string

5. Write three unit test methods using the unittest module. (4 Marks)

**Answer**

def identity(x):

    return x

class TestIdentity(unittest.TestCase):

    # Test 1: Simple value check

    def test\_basic\_value(self):

        self.assertEqual(identity(10), 10)

    # Test 2: String value check

    def test\_string\_value(self):

        self.assertEqual(identity("Kelly"), "Kelly")

    # Test 3: Failure check (tests if it handles different input/output)

    def test\_type\_mismatch(self):

        # Asserts that the result is NOT the boolean 'True'

        self.assertIsNot(identity(1), True)

6. Explain the evaluation strategy for function arguments. (2 Marks)

It is a strategy used in many programming languages were by arguments are evaluated fully before functions are called and the result of the evaluation is passed to the functions parameter and the changes made to the parameter inside the function do not affect the priginal variables out the function

**Answer**

7. Define programming language paradigm and list three major paradigms. (4

Marks)

**Answer**

A paradigm is a type of stye for programing

1 Imperative : getting a result by telling the computer what to do step by step and changing state

2 Object oriented (OOP) : Focuses on objects that bundle data and behavior

3 Funtional :focuses on mathematical functions and avoids state changes

8. Write a function that demonstrates the use of closures with a practical example. (4

Marks)

**Answer**

def make\_multiplier(factor):

    # 'factor' is the variable 'closed over'

    def multiplier(number):

        return number \* factor

    return multiplier # Returns the inner function

double = make\_multiplier(2)

triple = make\_multiplier(3)

print(f"Kelly's double: {double(5)}")  # 10

print(f"Kelly's triple: {triple(5)}") # 15

9. Distinguish between a syntax error and a semantic error, using code examples to

illustrate each. (4 Marks)

**Answer**

|  |  |  |
| --- | --- | --- |
|  | Syntax error | Semantic Error |
| Definition | Programming language rules are not followed (grammar rules) | When the code is correct but does the wrong thing or does not do what it meant to do |
| Response | The program wont even work | The program will run but is wrong |
| Example | x<10  # There are no colons | avg =(a/b)/2  # Used division instead of addition |

10. Write a lambda expression to find the square of a number and briefly explain the

concept of lambda calculus. (4 Marks)

**Answer**

square = lambda x: x \*\* 2

result = square (8)

print(f"The square for Kelly is {result}")

11. Define the concepts of lexical scope and dynamic scope in programming languages.

Give one example where the output would differ depending on the scoping rule. (5

Marks)

**Answer**

Lexical scope -is based on where the function is physically defined in the source code

Dynamic – based on the runtime call stack ( where the function was called from)

EXAMPLE

If variable x is global(10) and locally overridden in the called function (20) :

Lexical scope uses the global x(10)

Dynamic scope uses the local x (20 ) from the caller

12. List three differences between static typing and dynamic typing in programming

languages. (3 Marks)

**Answer**

|  |  |  |
| --- | --- | --- |
| Feature | Static type | Dynamic types |
| Check time | Early compile time | Late run time |
| Declaration | Types must be explicitly declared | Types are implicitly in inferred |
| Safety | Safer | Less safe |

13. List the Four Pillars of OOP and demonstrate each using class syntax. (6 Marks)

**Answer**

14. Write code showing pattern matching (using match-case or if-elif) for different

data types. (4 Marks)

**Answer**

def check data(data):

    match data:

        case str(s) if s == "Kelly":

            print("Found Kelly!")

        case int():

            print("It's an integer.")

        case list():

            print("It's a list.")

        case \_:

            print("Unknown type.")

check data("Kelly")

check\_data(10)

15. Create classes showing inheritance with \_\_init\_\_ methods and method overriding.

(6 Marks)

**Answer** class Person: # Parent Class

    def \_init\_(self, name):

        self.name = name

    def greeting(self):

        return f"Hello, I am {self.name}."

class Student(Person): # Child Class (Inherits from Person)

    def \_init\_(self, name, id):

        super().\_init(name) # Calls parent's \_\_init\_

        self.student\_id = id

    # Method Overriding: Replaces the parent's greeting

    def greeting(self):

        return f"Hi! I'm Student Kelly (ID: {self.student\_id})."

kelly = Student("Kelly", "K001")

print(kelly.greeting())

16. Compare lazy evaluation (generators) vs eager evaluation with code examples. (4

Marks)

**Answer**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Lazy evaluation** | **Eager evaluation** |
| Timing | Evaluated on demand | Evaluated immediately and fully |
|  |  |  |
| memory | Memory efficient | Memory intensive (stores the entire result) |

17. Explain the difference between a compiler and an interpreter. (3 Marks)

|  |  |  |
| --- | --- | --- |
|  | Compiler | Interpreter |
| Translation | Translates the enter source code into machine code before execution | Translates and excites the program line by line |
| Speed | Generally faster | Generally slow |

18. State three identifier naming rules with valid and invalid examples. (2 Marks)

​1 Start with a letter or an underscore (\\_).

​-Valid: kelly\_score, \_temp

​-Invalid: 2024\_data (starts with a digit)

​ 2 Cannot contain special characters (like!@\#\%^\&), except for the underscore (\\_).

​-Valid: Kelly name, user\_id

​-Invalid: kelly-name, price$

​ 3 Cannot be a reserved keyword (e.g., if, while, for, class).

​-Valid: if condition, for\_loop\_count

​-Invalid: class, def (reserved keywords

19. Explain the difference between compiled and interpreted programming languages.

Give one example of each. (6 Marks)

Answer

**Feature**

Compiled Languages (e.g., C++)

Interpreted Languages (e.g., Python)

**Process**

Use a compiler to create a standalone, optimized executable file once.

Use an interpreter to read and execute source code on the fly every time.

**Portability Lower.**

The executable is tied to the specific operating system and architecture. Higher. The same source code can run on any system with the correct interpreter.

**Use Case Systems programming**,

embedded systems, high-performance computing. Scripting, web applications, data science, rapid prototyping.

20. Write code demonstrating first-order functions (functions as arguments). (4 Marks)

# 1. A simple function to be passed as an argument

def append\_name(text):

    """Appends Kelly's name to a greeting."""

    return f"{text}, Kelly!"

# 2. A higher-order function that accepts another function as an argument

def apply\_greeting(func, greeting\_text):

    # The function 'func' is executed here using the given argument

    final\_message = func(greeting\_text)

    return final\_message

# Demonstration: Passing 'append\_name' (the function) to 'apply\_greeting'

message = apply\_greeting(append\_name, "Welcome back")

print(message)

# Output: Welcome back, Kelly!

21. Explain polymorphism in object-oriented programming using class examples. (6

Marks) class Person:

    """The base class."""

    def greeting(self):

        # Base implementation

        return "Hello, I am a standard person."

class Customer(Person):

    """A subclass that overrides the method for a specific behavior."""

    def greeting(self):

        # Specialized implementation for a customer

        return "Hello! I am a loyal customer and need help."

class Employee(Person):

    """A subclass demonstrating unique behavior for Kelly."""

    def greeting(self):

        # Specialized implementation for an Employee

        return "Good day! I'm Kelly and I'm here to assist you."

# Demonstrate Polymorphism: An object's type determines the method's behavior at runtime.

people = [Customer(), Employee()]

for person in people:

    print(person.greeting())

# Output:

# Hello! I am a loyal customer and need help.

# Good day! I'm Kelly and I'm here to assist you.

22. Define semantics and show code where syntax is correct but semantics are wrong. (4

Marks)^

Semantics is the study of the meaning of a program defining what the code does when executed

 # Syntactically Correct: This follows Python's rules perfectly.

customers served = 10

daily revenue = 500

# Semantically Wrong: If the intent is to calculate the average revenue per customer,

# the logic is flawed as the variables are swapped.

# It will calculate 10 / 500 = 0.02, which is an incorrect average.

average\_revenue = customers\_served / daily\_revenue

print(f"Kelly calculated the average as: ${average\_revenue:.2f}")

# The calculation (0.02) is wrong, demonstrating a semantic error.

23. Write three unit test methods using the unittest module. (4 Marks) #The code unit to be tested

def format\_name(first, last):

    """Puts first and last name together with proper casing."""

    return f"{first.capitalize()} {last.capitalize()}"

class TestNameFormatting(unittest.TestCase):

    # Method 1: Testing standard case conversion

    def test\_standard\_case(self):

        # We expect "Kelly Jones" from "kelly" and "jones"

        self.assertEqual(format\_name("kelly", "jones"), "Kelly Jones")

    # Method 2: Testing single-letter names

    def test\_single\_letter\_names(self):

        self.assertEqual(format\_name("k", "j"), "K J")

    # Method 3: Testing empty strings (edge case)

    def test\_empty\_string(self):

        # An empty name should produce just a space

        self.assertEqual(format\_name("", ""), " ")

# To run the tests (if this file were executed directly)

if \_name\_ == '\_main\_':

    # unittest.main()

    print("3 Unit Tests Defined.")

24. Explain the evaluation strategy for function arguments. (2 Marks)

Answer

The evaluation strategy dictates when and how the arguments passed to a function are computed before the function's body executes. The two primary strategies are:

​Call by Value: The value of the argument is computed, and a copy of that value is passed to the function. Any changes inside the function do not affect the original variable outside (standard for primitive types like numbers).

​Call by Reference: A reference (or memory address) to the original argument variable is passed. Any changes inside the function do affect the original variable outside (used for efficiency with large objects).

25. Compare recursive vs iterative solutions with code examples showing advantages of

each. (6 Marks)

Answer

1. Feature Recursive Solution Iterative Solution
2. Approach Function calls itself to solve sub-problems Uses loops to repeat operations
3. Readability Often more elegant and closer to mathematical logic More explicit and

sometimes more

efficient

1. Memory Usage Uses call stack; can lead to stack overflow Uses constant memory; more

space-efficient

1. Performance May be slower due to overhead of function calls Generally faster and

better for large inputs

example of recursive version:

def factorial\_recursive(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial\_recursive (n - 1)

print (factorial\_recursive (5)) # Output: 120

example of iterative version:

def factorial\_iterative(n):

result = 1

for i in range (2, n + 1):

result \*= i

return result

print (factorial\_iterative (5)) # Output: 120

26. Write a class demonstrating encapsulation using private attributes and getter/setter

methods. (5 Marks)

class Student:

def \_\_init\_\_ (self, name, age):

self. \_\_name = name # Private attribute

self. \_\_age = age # Private attribute

# Getter for name

def get\_name(self):

return self. \_\_name

# Setter for name

def set\_name (self, new\_name):

if isinstance (new\_name, str):

self. \_\_name = new\_name

# Getter for age

def get\_age(self):

return self. \_\_age

# Setter for age

def set\_age (self, new\_age):

if new\_age > 0:

self. \_\_age = new\_age

# Example usage

student = Student ("Kuda", 20)

print (student.get\_name ()) # Output: Kuda

student.set\_age (21)

print (student.get\_age ()) # Output: 21

a**nswe**r

27. Explain why version control is important for software projects. (3 Marks) Version control (like Git) is critical because it:

**Answer**

​1 Enables Collaboration:

Allows multiple developers (like a team including Kelly) to work on the same codebase simultaneously without overwriting each other's work, providing a mechanism for safely merging changes.

​2 Provides History & Auditing:

Records every change made, who made it, and why. This means you can trace the history of any line of code and revert to any stable state if a bug is introduced.

​3Manages Experimentation:

Facilitates branching, allowing developers to isolate new features or experimental code from the main (production) code, ensuring the main branch remains stable.

28. Write the complete Git workflow commands: initialize, add .py files, commit, and

push. (4 Marks)

**answer**

1 initialize the local repository in the projector folder

Git init

2 add all the new or modified phython files to the staging area

Git add \*.py

3 commit the staged changes to the local repository history

Git commit -m”Kelly:Finished

Initial implementation of function .”

4 push the local commit to the remote repository

Git out origin main

29. Create a simple DSL example for a specific domain (e.g., mathematical expressions).

(4 Marks)

**Answer**

# DSL classes for mathematical expressions

class Expr:

def evaluate(self):

pass

class Number(Expr):

def \_\_init\_\_ (self, value):

self.value = value

def evaluate(self):

return self.value

class Add(Expr):

def \_\_init\_\_ (self, left, right):

self. left = left

self. right = right

def evaluate(self):

return self. left. Evaluate () + self. right. Evaluate ()

class Multiply(Expr):

def \_\_init\_\_ (self, left, right):

self. left = left

self. right = right

def evaluate(self):

return self. left. Evaluate () \* self. right. Evaluate ()

# Example DSL usage

expr = Multiply (Add (Number (2), Number (3)), Number (4)) # (2 + 3) \* 4

print (expr. Evaluate ()) # Output: 20

30. Show how operator precedence works in the expression:

result = 2 + 3 \* 4 - 1 // 2 (2 Marks)

**ANSWER**

Operator precedence is the rule that dictates the order in which operators are evaluated. The order here is:

​\* and // (Multiplication and Floor Division)

​+ and - (Addition and Subtraction)

​Evaluation Steps:

Explanation :9+3\*4-1//8

1 Multiplication (\*) and Floor Division (//):

​9 + \mathbf{(3 \* 4)} - \mathbf{(1 // 8)}

​

​9 + 12 - 0

​ 2 Addition (+) and Subtraction (-) (evaluated left-to-right):

​\mathbf{(9 + 12)} - 0

​

​21 - 0

​3 Final Result:

​\mathbf{21}

31. Explain memory management including reference counting. (1 Mark)

**Answer**

It is the process by which a programming language or runtime system allocates, tracks and frees memory used by programs and one common technique is reference counting where each object keeps a count of how many references point to it.