**METHODIST COLLEGE OF ENGINEERING & TECHNOLOGY (An Autonomous Institution)**

**B.E. (ECE) III-Semester (AICTE) (Regular) Examination, Feb/March -2023**

**Subject: PYTHON PROGRAMMING**

**Time: 3 hours Max.Marks:60**

**Note: Missing data, if any, maybe suitably assumed.**

**PART-A**

**Answer All the questions.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q.No.** | **Questions** | **Marks** | **CO** | **BTL** |
| **1. a** | **The applications of Python are as follows**:  GUI based desktop applications  Image processing applications  Business and Enterprise applications  Prototyping  Web and web framework applications | **2** |  |  |
| **b** | # input  string = str(input())  # output  print(string) | **2** |  |  |
| **c** | the operator ==, which compares two operands and produces True if they are equal and False otherwise:  >>> 5 == 5 True  >>> 5 == 6 False | **2** |  |  |
| **d** | The break statement terminates the loop immediately and the control flows to the statement after the body of the loop.  The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop. | **2** |  |  |
| **e** | x is of type: <class 'int'>  y is of type: <class 'float'>  20.6  z is of type: <class 'float'> | **2** |  |  |
| **f** | |  |  | | --- | --- | | LIST | TUPLES | | Lists are mutable i.e they can be edited. | Tuples are immutable (tuples are lists which can’t be edited). | | Lists are slower than tuples. | Tuples are faster than list. | | Syntax: list\_1 = [10, ‘Chelsea’, 20] | Syntax: tup\_1 = (10, ‘Chelsea’ , 20) | | **2** |  |  |
| **g** | def add\_numbers(x,y):  sum = x + y  return sum  print("The sum is", add\_numbers(5, 20))  Output:  The sum is 25 | **2** |  |  |
| **h** | Python Local Variables  When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.  def f():  # local variable  s = "Hello"  print(s)  f()  output:  Hello | **2** |  |  |
| **i** | The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in a module | **2** |  |  |
| **j** | a. IndexError: list index out of range  b. KeyError: what  c. IOError: [Errno 2] No such file or directory: 'filename' | **2** |  |  |

**PART-B**

**Answer Any Five questions**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Q.No.** |  | **Questions** | **Marks** | **CO** | **BTL** |
| **2.** | **a** | An algorithm is a well-defined computational procedure that takes some values or the set of values, as an input and produces a set of values or some values, as an output.   * The algorithm improves the efficiency of an existing technique. * To compare the performance of the algorithm with respect to other techniques. * The algorithm gives a strong description of requirements and goal of the problems to the designer. * The algorithm provides a reasonable understanding of the flow of the program. * The algorithm measures the performance of the methods in different cases (Best cases, worst cases, average cases). * The algorithm identifies the resources (input/output, memory) cycles required by the algorithm. * With the help of an algorithm, we can measure and analyze the complexity time and space of the problems. * The algorithm also reduces the cost of design.   **step 1** − START  **step 2** − declare three integers **a**, **b** & **c**  **step 3** − define values of **a** & **b**  **step 4** − add values of **a** & **b**  **step 5** − store output of step 4 to **c**  **step 6** − print **c**  **step 7** − STOP | **4** |  |  |
| **b** | * Python is an interpreted language, so it doesn’t need to be compiled before execution, unlike [languages such as C](https://intellipaat.com/blog/tutorial/c-tutorial/). * Python is dynamically typed, so there is no need to declare a variable with the data type. Python Interpreter will identify the data type on the basis of the value of the variable.   For example, in Python, the following code line will run without any error:  a = 100  a = "Intellipaat"   * Python follows an **object-oriented programming** paradigm with the exception of having access specifiers. Other than access specifiers (public and private keywords), Python has classes, inheritance, and all other usual OOPs concepts. * Python is a **cross-platform language**, i.e., a Python program written on a Windows system will also run on a Linux system with little or no modifications at all. * Python is literally a **general-purpose language**, i.e., Python finds its way in various domains such as web application development, automation, Data Science, Machine Learning, and more. | **4** |  |  |
| **3.** | **a** | Chained Conditionals  Sometimes there are more than two possibilities and we need more than two  branches. One way to express a computation like that is a **chained conditional**:  if x < y:  print('x is less than y')  elif x > y:  print('x is greater than y')  else:  print('x and y are equal')  elif is an abbreviation of “else if ”. Again, exactly one branch will run. There is no  limit on the number of elif statements. If there is an else clause, it has to be at the  end, but there doesn’t have to be one.  if choice == 'a':  draw\_a()  elif choice == 'b':  draw\_b()  elif choice == 'c':  draw\_c()  Each condition is checked in order. If the first is false, the next is checked, and so on.  If one of them is true, the corresponding branch runs and the statement ends. Even if  more than one condition is true, only the first true branch runs.  Nested Conditionals  One conditional can also be nested within another. We could have written the example  in the previous section like this:  if x == y:  print('x and y are equal')  else:  if x < y:  print('x is less than y')  else:  print('x is greater than y')  The outer conditional contains two branches. The second branch contains another if statement, which has two  branches of its own. Those two branches could contain conditional statements as well.  Although the indentation of the statements makes the structure apparent, nested conditionals very quickly become very  difficult to read. In general, it is a good idea to avoid them when we can. | **4** |  |  |
| **b** | When an expression contains more than one operator, the order of evaluation  depends on the **order of operations**. For mathematical operators, Python follows  mathematical convention. The acronym **PEMDAS** is a useful way to remember the  rules:  • **P**arentheses have the highest precedence and can be used to force an expression  to evaluate in the order you want. Since expressions in parentheses are evaluated  first, 2 \* (3-1) is 4, and (1+1)\*\*(5-2) is 8. You can also use parentheses to  make an expression easier to read, as in (minute \* 100) / 60, even if it doesn’t  change the result.  • **E**xponentiation has the next highest precedence, so 1 + 2\*\*3 is 9, not 27, and 2  \* 3\*\*2 is 18, not 36.  **M**ultiplication and **D**ivision have higher precedence than **A**ddition and **S**ubtraction.  So 2\*3-1 is 5, not 4, and 6+4/2 is 8, not 5.  • Operators with the same precedence are evaluated from left to right (except  exponentiation). So in the expression degrees / 2 \* pi, the division happens  first and the result is multiplied by pi. To divide by 2*π*, you can use parentheses  or write degrees / 2 / pi. | **4** |  |  |
| **4.** | **a** | **# using = operator**  str1 = "Hello, world!"  str2 = "I love Python."  str3 = "Hello, world!"  print(str1 == str2)  print(str1 == str3)  Output: False  True  **# using + operator**  greet = "Hello, "  name = "Jack"  result = greet + name  print(result)  Output: Hello, Jack  **# using len operator**  greet = 'Hello'  **# count length of greet string**  print(len(greet))  **# Output: 5**  print('a' in 'program') # True  print('at' not in 'battle') False | **4** |  |  |
| **b** | Python provides you with a much-needed functionality of converting one form of data type into the needed one and this is known as type conversion.  **Type Conversion is classified into types:**  1. Implicit Type Conversion: In this form of [**type conversion python**](https://intellipaat.com/blog/tutorial/python-tutorial/type-conversion-in-python/) interpreter helps in automatically converting the data type into another data type without any User involvement.  2. Explicit Type Conversion: In this form of Type conversion the data type inn changed into a required type by the user.  Various Functions of explicit conversion are shown below:  int() – function converts any data type into integer.  float() – function converts any data type into float.  ord() – function returns an integer representing the Unicode character  hex() – function converts integers to hexadecimal strings.  oct() – function converts integer to octal strings.  tuple() – function convert to a tuple.  set() – function returns the type after converting to set.list() – function converts any data type to a list type.  dict() – function is used to convert a tuple of order (key,value) into a dictionary.  str() – function used to convert integer into a string.  complex(real,imag) – function used to convert real numbers to complex(real,imag) numbers | **4** |  |  |
| **5.** | **a** | In Python, an anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.  Python lambda Syntax:  lambda arguments : expression  Example:  calc = lambda num: "Even number" if num % 2 == 0 else "Odd number"  print(calc(20))  Difference between lambda and normal function call  The main difference between lambda function and other functions defined using def keyword is that, we cannot use multiple statements inside a lambda function and allowed statements are also very limited inside lambda statements. Using lambda functions to do complex operations may affect the readability of the code. | **4** |  |  |
| **b** | def recur\_fibo(n):  if n <= 1:  return n  else:  return(recur\_fibo(n-1) + recur\_fibo(n-2))  # take input from the user  nterms = int(input("How many terms? "))  # check if the number of terms is valid  if nterms <= 0:  print("Plese enter a positive integer")  else:  print("Fibonacci sequence:")  for i in range(nterms):  print(recur\_fibo(i)) | **4** |  |  |
| **6.** | **a** | def longest\_word(filename):  with open(filename, 'r') as infile:  words = infile.read().split()  max\_len = len(max(words, key=len))  return [word for word in words if len(word) == max\_len]  print(longest\_word('test.txt')) | **4** |  |  |
| **b** | **Python try...except Block**  The try...except block is used to handle exceptions in Python. Here's the syntax of try...except block:  **try:**  **# code that may cause exception**  **except:**  **# code to run when exception occurs**  Here, we have placed the code that might generate an exception inside the try block. Every try block is followed by an except block.  When an exception occurs, it is caught by the except block. The except block cannot be used without the try block. | **4** |  |  |
| **7.** | **a** |  | **4** |  |  |
| b | year=int(input("Enter year to be checked:"))  if(year%4==0 and year%100!=0 or year%400==0):  print("The year is a leap year!")  else:  print("The year isn't a leap year!") | 4 |  |  |
| 8. | a | List and Tuple objects are sequences. A dictionary is a hash table of key-value pairs. List and tuple is an ordered collection of items. Dictionary is unordered collection.  List and dictionary objects are mutable i.e. it is possible to add new item or delete and item from it. Tuple is an immutable object. Addition or deletion operations are not possible on tuple object.  Each of them is a collection of comma-separated items. List items are enclosed in square brackets [], tuple items in round brackets or parentheses (), and dictionary items in curly brackets {}  >>> L1=[12, "Ravi", "B.Com FY", 78.50] #list  >>> T1=(12, "Ravi", "B.Com FY", 78.50)#tuple  >>> D1={"Rollno":12, "class":"B.com FY", "precentage":78.50}#dictionary  List and tuple items are indexed. Slice operator allows item of certain index to be accessed  >>> print (L1[2])  B.Com FY  >>> print (T1[2])  B.Com FY  Items in dictionary are not indexed. Value associated with a certain key is obtained by putting in square bracket. The get() method of dictionary also returns associated value.  >>> print (D1['class'])  B.com FY  >>> print (D1.get('class'))  B.com FY | 4 |  |  |
| b | The import statement is the most common way of invoking the import machinery, but it is not the only way.  import module\_name  When the import is used, it searches for the module initially in the local scope by calling \_\_import\_\_() function. The value returned by the function is then reflected in the output of the initial code.  **import module\_name.member\_name**  In the above code module, math is imported, and its variables can be accessed by considering it to be a class and pi as its object.  The value of pi is returned by \_\_import\_\_(). pi as a whole can be imported into our initial code, rather than importing the whole module.  **from module\_name import \***  In the above code module, math is not imported, rather just pi has been imported as a variable.  All the functions and constants can be imported using \*. | 4 |  |  |
| 9. | a | **os.path.basename(path) :** It is used to return the basename of the file . This function basically return the file name from the path given.  **os.path.dirname(path) :** It is used to return the directory name from the path given. This function returns the name from the path except the path name.  **os.path.isabs(path) :** It specifies whether the path is absolute or not. In Unix system absolute path means path begins with the slash(‘/’) and in Windows that it begins with a (back)slash after chopping off a potential drive letter.  **os.path.isdir(path) :** This function specifies whether the path is existing directory or not. | 4 |  |  |
| b | # Define two matrix A and B in program  A = [[5, 4, 3],  [2, 4, 6],  [4, 7, 9]]  B = [[3, 2, 4],  [4, 3, 6],  [2, 7, 5]]  # Define an empty matrix to store multiplication result  multiResult = [[0, 0, 0],  [0, 0, 0],  [0, 0, 0]]  # Using nested for loop method on A & B matrix  for m in range(len(A)):  for n in range(len(B[0])):  for o in range(len(B)):  multiResult[m][n] += A[m][o] \* B[o][n] # Storing multiplication result in empty matrix  # Printing multiplication result in the output  print("The multiplication result of matrix A and B is: ")  for res in multiResult:  print(res) | 4 |  |  |

**\*\*\*\*\*\***