

Sample Question Paper-C

1. (a) Mozilla Firefox. (1)
- (b) 1 Tera Byte= 0.000976562 PetaByte (1)
- (c) Lesser number of instructions in a program (1)

(d)	PROM	EPROM
1. Once written, data cannot be erased from it.	Data can be erased by electrical or magnetic means.	
2. Data is not disrupted by any electrical interference	Data is disrupted by electrical interference.	

- (2)
2. (a) 1 GHz= 1000 MHz (1)
- (b) Unique Universal and Uniform character Encoding . (1)
- (c) (i) Management of System Hardware (2)
- (ii) Management of System memory (2)
- (d) (i) Top Down Program Development (2)
- (ii) No use of modules in the program (2)
3. (a) Scope can be defined as the part of program where the lifetime of variable exists. (1)

(b) len()

>>> len(a) # get number of characters in the string

5

capitalize()

The method capitalize() returns a copy of the string with only its first character capitalized. For 8-bit strings, this method is locale-dependent.

Syntax

str.capitalize()

Example

str = "this is string example....wow!!!";

print "str.capitalize() : ", str.capitalize()

result:

str.capitalize() : This is string example....wow!!!

(2)

(c) **def** rot(x):

x=5

print randint(x,8) may return numbers between 5 and 8

(2)

(d) **ALGORITHM:**

STEP 1 : START

STEP 2 : Input sides of triangle

STEP 3 : Find largest of three sides

STEP 4 : If (Largest Side)= (Side1)+(Side 2)GOTO STEP 5 ELSE STEP 6

STEP 5 : Print "Right Angled Triangle" GOTO STEP 7

STEP 6 : Print "Not a Right Angled Triangle." GOTO STEP 7

STEP 7 : END

(2)

(e) **String Operators**

(i) **+ Operator**

For strings, + means “concatenate”

Example:

```
>>> a = "hello"           # using double quote
>>> b = 'world'          # using single quote
>>> a + b                 # concatenates string
'helloworld'
```

(ii) *** Operator**

This Operator creates new strings, concatenating multiple copies of the same string .

Example

```
>>>a=hello
>>>a*2
HelloHello
```

(iii) **in**

Evaluates to true if it finds a variable in the specified sequence and false otherwise
x in y, here in results in a 1 if x is a member of sequence y.

(iv) **not in**

Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.
x not in y, here not in results in a 1 if x is not a member of sequence y.

(v) **[m:n] Slice Operator**

Gives the character from the given index

Like 119 people like this. Be the f irst of your f riends.

Example

```
>>> a = "hello"
>>> a[0]    # get first character (same as list, zero indexed)
'h'
>>> a[-1]   # get last character
'o'
>>> a[0:2]  # get first two characters (4)
```

4. (a) DBMS is a collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMSs, ranging from small systems that run on personal computers to huge systems that run on mainframes. (1)

- (b) len() finds the length of the list.

reverse() reverses the list.

Example:

```
l=[1,2,3,4]
l.len() returns 4
l.reverse returns 4,3,2,1 (2)
```

- (c) The Python interpreter can be used in two modes: interactive and scripted. In *interactive* mode, Python responds to each statement while we type. In *script* mode, we give Python a file of statements and turn it loose to interpret all of the statements in that script. Both modes produce identical results. When we're producing a finished application program, we set it up to run as a script.

Interactive Mode:

It is a command line shell which gives immediate output for each statement.

>>> is the representation of each new line in Interactive mode. It is a better mode than script mode since the output is faster.

Example:

```
>>> def abc(x):
>>> x=input()
>>> print x
```

Script Mode:

In script mode, we have to write a program in an editor. It is not a console based system. The difference here is that we have to write and save the whole program at one and then call it from the python shell. It is slower as compared to interactive mode but it is better when writing big programs. (3)

- (d) The `map()` function applies a function to every member of a list. Here a list of numbers 32 to 255 is changed to a list of the corresponding ASCII characters. Then `map()` is used in a more complex application to combine the two lists to form a dictionary containing number:character pairs.

Syntax:

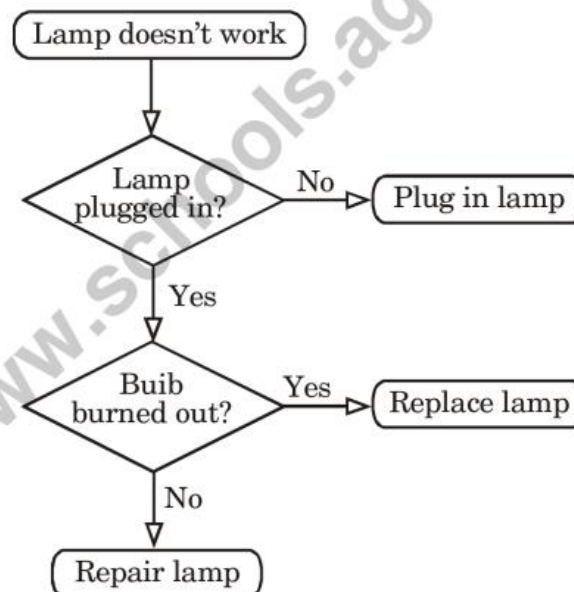
`map(f, iterable)`

Example:

```
map(operator.setitem, [charD]*len(keyList), keyList,
valueList)
```

(3)

- (e) A **flowchart** is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution to a given problem. Process operations are represented in these boxes, and arrows; rather, they are implied by the sequencing of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.



(3)

- (f) (i) 30,10,18
(ii) No Output, size of a is less than the index.
(iii) 5,10,18
(iv) 30,20,18

(4)

5. (a) A **compiler** is a computer program (or set of programs) that transforms source code written in a programming language (the *source language*) into another computer language (the *target language*) (1)

- (b) Basically there are two types of microprocessor architectures:

(i) **RISC** : RISC (reduced instruction set computer) is a microprocessor that is designed to perform a smaller number of types of computer instructions so that it can operate at a

higher speed (perform more millions of instructions per second, or MIPS). Since each instruction type that a computer must perform requires additional transistors and circuitry, a larger list or set of computer instructions tends to make the microprocessor more complicated and slower in operation.

(ii) **CISC** : The term “CISC” (complex instruction set computer or computing) refers to computers designed with a full set of computer instructions that were intended to provide needed capabilities in the most efficient way. (3)

(c) happyhappy

happyhappyhappy

happyhappyhappyhappy (3)

(d) def log(x):

sum = 0.0

n = 0.0

term = 1.0

while (term > .0000000001):

#loops until the iterations grow so large that ‘term’ becomes negligibly small

term = ((x ** (2 * n - 1.0))) / (factorial(n + 1.0))

if n % 2 == 0:

sum += term

else:

sum -= term

n += 1

return sum (4)

6. (a) a function call is a statement in which a function is invoked from another function (1)

(b) from math import pi

r = 123

area = pi * r**2

print “Area of circle with radius 123 = “ + str(area) (2)

(c) 0

1

2 (3)

(d) The **string** module contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings. In addition, Python’s built-in string classes support the sequence type methods described in the *Sequence Types* — *str*, *unicode*, *list*, *tuple*, *bytearray*, *buffer*, *xrange* .

String constants

The constants defined in this module are:

string.ascii_letters

The concatenation of the **ascii_lowercase** and **ascii_uppercase** constants described below. This value is not locale-dependent.

string.ascii_lowercase

The lowercase letters ‘abcdefghijklmnopqrstuvwxyz’. This value is not locale-dependent and will not change.

string.ascii_uppercase

The uppercase letters ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’. This value is not locale-dependent and will not change.

string.digits

The string ‘0123456789’.

String Formatting

class string.Formatter

The **Formatter** class has the following public methods:

format(*format_string*, **args*, ***kwargs*)¶

format() is the primary API method. It takes a format string and an arbitrary set of positional and keyword arguments. **format()** is just a wrapper that calls **vformat()**

(4)

7. (a) Python is an interpreted language.

(1)

(b) Soft Real Time Systems are the systems where there is not a guarantee that the output will be produced in a stipulated time.

(1)

(c) **def** possible_anagrams(word):

"given a word, return a list of possible anagrams for that word"

if len(word) == 1:

yield word

for i, letter **in** enumerate(word):

recursively call with a word whose characters have been rotated

for s **in** possible_anagrams(word[i+1:] + word[:i]):

yield '%s%s' % (letter, s)

def anagrams(word):

"given a word, return a list of anagrams in an english dictionary"

 words = [w.rstrip() **for** w **in** open('WORD.LST')]

return [a **for** a **in** possible_anagrams(word) **if** a **in** words]

if __name__ == "__main__":

print anagrams('python')

(4)

(d) **Fourth Generation of Computers (1972-1984)**

In this generation, there were developments of large-scale integration or LSI (1000 devices per chip) and very large-scale integration or VLSI (10000 devices per chip). These developments enabled the entire processor to fit into a single chip and in fact, for simple systems, the entire computer with processor; main memory and I/O controllers could fit on a single chip.

Core memories now were replaced by semiconductor memories and high-speed vectors dominated the scenario. Names of few such vectors were Cray1, Cray X-MP and Cyber205. A variety of parallel architectures developed too, but they were mostly in the experimental stage.

As far as programming languages are concerned, there were development of high-level languages like FP or functional programming and PROLOG (programming in logic). Declarative programming style was the basis of these languages where a programmer could leave many details to the compiler or runtime system. Alternatively languages like PASCAL, C used imperative style. Two other conspicuous developments of this era were the C programming language and UNIX operating system. Ritchie, the writer of C and Thompson together used C to write a particular type of UNIX for DEC PDP 11. This C based UNIX was then widely used in many computers.

(4)



