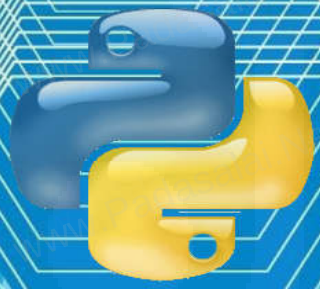
DELIGHTERS GUIDE

12

COMPUTER SCIENCE



1,2,3 & 5 Marks

STUDY MATERIAL

Prepared By

T.THIRUMALAI, M.Sc(CS).,B.Ed.,

UNIT-I

CHAPTER – 1 FUNCTION

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

CHOOSE THE CORRECT ANSWER:

1. Which of the following is important criteria complete the task?
   a) Program b) code c) **algorithm** d) pseudo code
2. The duration of computation time must be independent of
   a) Compiler b) pseudo code
   c) Programming language d) **a & c**
3. The algorithms are expressed using _____of a programming language.
   a) Functions b) subroutines c) **statements** d) reference
4. If a bulk of statements to be repeated for many no.of times then _____are used to finish the task.
   a) **subroutines** b) programs c) required d) statements
5. _____ is the basic building blocks of computer programs.
   a) Programs b) function c) pure function d) **Subroutines**
6. _____ are small sections of code that are used to perform a particular task that can be used repeatedly.
   a) Impure function b) **subroutines**
   c) Pure function d) programming language
7. In Programming languages these subroutines are called as _____.
   a) Subroutines b) **Functions** c) pseudo code d) Inference
8. A _____ is a unit of code that is often defined within a greater code structure.
   a) Routine b) Node c) **Function** d) Program
9. _____ is distinct syntactic blocks.
   a) **Definitions** b) Declaration c) Statement d) Required
10. _____ is the variables in a function definition.
    a) Algorithms b) programs c) **Parameters** d) Functions
11. _____ is the values which are passed to a function definition.
    a) **Arguments** b) impurefunction c) statement d) algorithm
12. There are _____ types of parameters are in the functions.
    a) 4 b) **2** c) 3 d) 5
13. The syntax for function definition is _____.
    a) **let rec fn a1 a2 ... an := k** b) let receive fn a1 a2 ... an := k
    c) let rec function a1 a2 ... an := k d) let rec fn a1 a2 ... an :> k
14. The keyword ____ is required if 'fn' is to be a recursive function; otherwise it may be omitted.
    a) record b) receive c) **rec** d) recover
15. A function definition which call itself is called _____ function.
    a) record b) **recursive**
    c) return d) destroy
16. All functions are _____ definitions.
    a) single b) dynamic c) **static** d) dual
17. An _____ is a set of action that an object can do.
    a) interconnect b) interflow c) function d) **interface**

18. _____ are functions which will give exact result when the same arguments are passed.
    a) **Pure functions** b) inner function
    c) outer function d) impure function
19. The another name of side - effect _____.
    a) Pure functions b) inner function
    c) outer function d) **impure function**
20. The return value of the _____ solely depends on its arguments passed.
    a) **Pure functions** b) inner function
    c) outer function d) impure function
21. _____ contains a set of code that works on many kinds of inputs and produces a concrete output.
    a) interconnect b) interflow c) **function** d) interface
22. When you write the type annotation the _____ are mandatory in the function definition.
    a) set braces b) **parentheses** c) slashes d) dots
23. _____ carries out the instructions defined in the interface.
    a) Abstraction b) Reduction c) Collusion d) **Implementation**
24. There are _____ characteristics have in interface.
    a) 3 b) 4 c) 5 d) **2**
25. ***let rec fna1 a2 ... an := k*** in this ***'fn'*** indicating the _____ of the function name.
    a) String b) Character c) **Identifier** d) Constant
26. _____ do not modify the arguments which are passed to them.
    a) inner function b) **Pure functions**
    c) outer function d) impure function
27. _____ may modify the arguments which are passed to them.
    a) inner function b) Pure functions
    c) outer function d) **Impure function**
28. One of the most popular groups of side effects is modifying the variable _____ of function.
    a) **outside** b) topside c) inside d) None of these
29. ***let y: = 0***
    (int) inc (int) x
    y: = y + x;
    return (y)
    In the above Algorithm function. The side effects of the _____ function is it is changing the data of the external visible variable ___ .
    a) gcd( ), x b) int( ), y c) **inc (), y** d) inc (), x
30. An object's _____ and _____ is controlled by sending functions to the object.
    a) **attributes, behaviour** b) function, behaviour
    c) function, attributes d) None of these

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

2-Marks

1. What is subroutine?

Subroutines are the basic building blocks of computer programs. Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly.

2. Define Algorithm.

Algorithms are expressed using statements of a programming language.

3. Define Function with respect to Programming language.

A function is a unit of code that is often defined within a greater code structure. Specifically, a function contains a set of code that works on many kinds of inputs, like variants, expressions and produces a concrete output.

4. Write the inference you get from X:=(78).

In the above function defnition if expression can return ***1*** in the then branch, by the ***typing*** rule the entire if expression has type ***int***. We get inference of expression is **int**.

5. Differentiate interface and implementation.

Interface	Implementation
Interface just defines what an object can do, but won't actually do it.	Implementation carries out the instructions defined in the interface.

6.

Which of the following is a normal function definition and which is recursive function defnition.

i) let rec sum x y:
return x + y
ii) let disp :
print 'welcome'
iii) let rec sum num:
if (num!=0) then return num + sum (num-1)
else
return num

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

(i) Recursive function definition (ii) Normal function definition
(iii) Recursive function definition

3-Marks

7. Mention the Characteristics of interface.

Characteristics of interface

• The class template specifies the interfaces to enable an object to be created and operated properly.

• An object's attributes and behavior is controlled by sending functions to the object.

8. Why strlen is called pure function?

strlen is a pure function because the function takes one variable as a parameter, and accesses it to find its length. This function reads external memory but does not change it, and the value returned derives from the external memory accessed.

9. What is the side effect function of impure function? Give example.

The variables used inside the function may cause side effects though the functions which are not passed with any arguments. In such cases the function is called impure function.

For example

The mathematical function random( ) will give different outputs for the same function call.

```
let Random number
let a := random()
if a > 10 then
return: a
else
return: 10
```

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

10. Differentiate between Pure function and Impure function.

Pure Function	Impure Function
The return value of the pure functions solely depends on its arguments passed. Hence, if you call the pure functions with the same set of arguments, you will always get the same return values. They do not have any side effects.	The return value of the impure functions does not solely depend on its arguments passed. Hence, if you call the impure functions with the same set of arguments, you might get the different return values. For example, random(), Date().
They do not modify the arguments which are passed to them.	They may modify the arguments which are passed to them.

11. What happens if you modify a variable outside the function? Give an example.

Modify variable outside a function

One of the most popular groups of side effects is modifying the variable outside of function. For example

```
let y: = 0
(int) inc (int) x
y: = y + x;
return (y)
```

In the above example the value of y get changed inside the function defnition due to which the result will change each time. The side effects of the inc () function is it is changing the data of the external visible variable **'y'**. As you can see some side effects are quite easy to spot and some of them may tricky. A good sign that our function impure *(has side effects)* is that it doesn't take any arguments and it doesn't return any value.

5 - Marks

12. What are called Parameters and write a note on
(i) Parameter without Type (ii) Parameter with Type

Parameters (and arguments)

Parameters are the variables in a function definition and arguments are the values which are passed to a function definition.

1. Parameter without Type

Let us see an example of a function definition:

```
(requires: b>=0 )
(returns: a to the power of b) let rec pow a b:=
        if b=0 then 1
        else a * pow a (b-1)
```

In the above function definition variable *'b'* is the parameter and the value which is passed to the variable *'b'* is the argument. Te precondition ***(requires)*** and post condition ***(returns)*** of the function is given. Note we have not mentioned any types: ***(data types)***. Some language compiler solves this type ***(data type)*** inference problem algorithmically, but some require the type to be mentioned.

In the above function definition if expression can return ***1*** in the then branch, by the ***typing*** rule the entire if expression has type ***int***. Since the if expression has type ***'int',*** the function's return type also be *'int'*. ***'b'*** is compared to *0* with the equality operator, so ***'b'*** is also a type of *'int'*. Since ***'a'*** is multiplied with another expression using the * operator, ***'a'*** must be an int.

2. Parameter with Type

Now let us write the same function definition with types for some reason:

```
(requires: b> 0 )
(returns: a to the power of b )
    let rec pow (a: int) (b: int) : int :=
        if b=0 then 1
    else a * pow b (a-1)
```

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

When we write the type annotations for ***'a'*** and ***'b'*** the parentheses are mandatory. Generally we can leave out these annotations, because it's simpler to let the compiler infer them. There are times we may want to explicitly write down types. This is useful on times when you get a type error from the compiler that doesn't make sense. Explicitly annotating the types can help with debugging such an error message. The syntax to define functions is close to the mathematical usage: the definition is introduced by the keyword ***let***, followed by the ***name*** of the function and its *arguments*; then the formula that computes the image of the argument is written after an = sign. If you want to define a recursive function: use ***"let rec" instead of "let".***

Syntax:

The syntax for function definitions:

```
let rec fna1 a2 ... an := k
```

Here the ***'fn'*** is a variable indicating an identifier being used as a function name. The names ***'a1'*** to ***'an'*** are variables indicating the identifiers used as parameters. The keyword ***'rec'*** is required if ***'fn'*** is to be a recursive function; otherwise it may be omitted.

13. Identify in the following program

let rec gcd a b :=
if b <> 0 then gcd b (a mod b) else return a

i) Name of the function
ii) Identify the statement which tells it is a recursive function
iii) Name of the argument variable
iv) Statement which invoke the function recursively
v) Statement which terminates the recursion

(i) gcd( ) function
(ii) recursively called till the variable *'b'* becomes *'0'*
(iii) *b* and *(a mod b)* are two arguments passed to *'a'* and *'b'* of the gcd function.
(iv) *(a mod b) until 'b' became '0'.*
(v) return a. or (When variable 'b' became '0' terminated).

14. Explain with example Pure and impure functions.

PURE FUNCTIONS

Pure functions are functions which will give exact result when the same arguments are passed. For example the mathematical function sin (0) always results **0**. This means that every time you call the function with the same arguments, you will always get the same result. A function can be a pure function provided it should not have any external variable which will alter the behaviour of that variable. Let us see an example

let square x
return: x * x

T.THIRUMALAI, M.SC(CS)., B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

The above function square is a pure function because it will not give different results for same input. There are various theoretical advantages of having pure functions. One advantage is that if a function is pure, then if it is called several times with the same arguments, the compiler only needs to actually call the function once. Let's see an example

let i: = 0;
if i <strlen (s) then
-- Do something which doesn't affects
++i

If it is compiled, ***strlen (s)*** is called each time and strlen needs to iterate over the whole of ***'s'.*** If the compiler is smart enough to work out that strlen is a pure function and that ***'s'*** is not updated in the loop, then it can remove the redundant extra calls to strlen and make the loop to execute only one time. From these what we can understand, strlen is a pure function because the function takes one variable as a parameter, and accesses it to find its length. This function reads external memory but does not change it, and the value returned derives from the external memory accessed.

IMPURE FUNCTIONS

The variables used inside the function may cause side effects though the functions which are not passed with any arguments. In such cases the function is called impure function. When a function depends on variables or functions outside of its defnition block, you can never be sure that the function will behave the same every time it's called. For example the mathematical function random() will give different outputs for the same function call.

```
let Random number
let a := random()
if a > 10 then
return: a
else
return: 10
```

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

Here the function Random is impure as it is not sure what will be the result when we call the function.

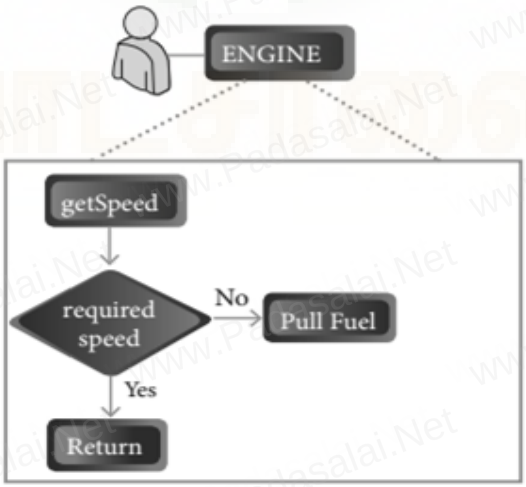
15. Explain with an example interface and implementation.

INTERFACE VS IMPLEMENTATION

An interface is a set of action that an object can do. For example when you press a light switch, the light goes on, you may not have cared how it splashed the light. In Object Oriented Programming language, an Interface is a description of all functions that a class must have in order to be a new interface. In our example, anything that ***"ACTS LIKE"*** a light, should have function definitions like turn_on () and a turn_off (). The purpose of interfaces is to allow the computer to enforce the properties of the class of ***TYPE T*** *(whatever the interface is)* must have functions called *X, Y, Z,* etc.

A class declaration combines the external interface *(its local state)* with an implementation of that interface *(the code that carries out the behaviour).* An object is an instance created from the class. The interface defines an object's visibility to the outside world.

In object oriented programs classes are the interface and how the object is processed and executed is the implementation.



The person who drives the car doesn't care about the internal working. To increase the speed of the car he just presses the accelerator to get the desired behaviour. Here the accelerator is the interface between the driver *(the calling / invoking object)* and the engine *(the called object)*.

In this case, the function call would be Speed (70): This is the interface. Internally, the engine of the car is doing all the things. It's where fuel, air, pressure, and electricity come together to create the power to move the vehicle. All of these actions are separated from the driver, who just wants to go faster. Thus we separate interface from implementation. Let us see a simple example, consider the following implementation of a function that finds the minimum of its three arguments:

```
let min 3 x y z :=
        if x < y then
                if x < z then x else z
        else
                if y < z then y else z
```

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

Education is End of the Air.
Learning is Your Breath End.
Breath! Love the Breath!!
--- *Mr.Hill.*

T.THIRUMALAI, M.SC(CS).,B.ED.,
Cell: 9750827717, 7010154722
thirumalaibca.46@gmail.com

Learn – Teach

Never
Stop to Learning
Because
Life never stops
Teaching.