J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com
Puducherry.

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com
Puducherry.

(05-07-2019)

COMPUTER SCIENCE

1. FUNCTIONS

Section – A

Choose the best answer **(1 Mark)**

1. The small sections of code that are used to perform a particular task is called
   (A) Subroutines (B) Files (C) Pseudo code (D) Modules
2. Which of the following is a unit of code that is often defined within a greater code structure?
   (A) Subroutines **(B) Function** (C) Files (D) Modules
3. Which of the following is a distinct syntactic block?
   (A) Subroutines (B) Function **(C) Definition** (D) Modules
4. The variables in a function definition are called as
   (A) Subroutines (B) Function (C) Definition **(D) Parameters**
5. The values which are passed to a function definition are called
   (A) Arguments (B) Subroutines (C) Function (D) Definition
6. Which of the following are mandatory to write the type annotations in the function definition?
   (A) Curly braces **(B) Parentheses** (C) Square brackets (D) indentations
7. Which of the following defines what an object can do?
   (A) Operating System (B) Compiler **(C) Interface** (D) Interpreter
8. Which of the following carries out the instructions defined in the interface?
   (A) Operating System (B) Compiler **(C) Implementation** (D) Interpreter
9. The functions which will give exact result when same arguments are passed are called
   (A) Impure functions (B) Partial Functions
   (C) Dynamic Functions **(D) Pure functions**
10. The functions which cause side effects to the arguments passed are called
    (A) Impure functions (B) Partial Functions
    (C) Dynamic Functions (D) Pure functions

Section-B

Answer the following questions **(2 Marks)**

1. What is a subroutine?

- Subroutines are small sections of code that are used to perform a particular task that can be used repeatedly.

2. Define Function with respect to Programming language.

- A function is a unit of code that is often defined within a greater code structure.
- A function works on many kinds of inputs and produces a concrete output

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

- **X:=(78)** is a function definition.
- Definitions bind values to names.
- Hence, the value **78** bound to the name **'X'**.

4. 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

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

i) let rec sum x y:
return x + y

Ans: Recursive Function

ii) let disp :
print 'welcome'

Ans: Normal Function

iii) let rec sum num:
if (num!=0) then return num + sum (num-1)
else
return num

Ans: Recursive Function

Section-C

Answer the following questions **(3 Marks)**

1. Mention the characteristics of Interface.

- The class template specifies the interfaces to enable an object to be created and operated properly.
- An object's attributes and behaviour is controlled by sending functions to the object.

2. 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.

3. What is the side effect of impure function. Give example.

➢ Impure Function has the following side effects,

- Function impure *(has side effect)* is that it doesn't take any arguments and it doesn't return any value.
- Function depends on variables or functions outside of its definition block.
- It never assure you that the function will behave the same every time it's called.

- **Example:**

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

- Here, the result of ***inc()*** will change every time if the value of ***'y'*** get changed inside the function definition.
- Hence, the side effect of inc () function is changing the data of the external variable ***'y'.***

4. Differentiate pure and impure function.

Pure Function	Impure Function
• Pure functions will give exact result when the same arguments are passed.	• Impure functions never assure you that the function will behave the same every time it's called.
• Pure function does not cause any side effects to its output.	• Impure function causes side effects to its output.
• The return value of the pure functions solely depends on its arguments passed.	• The return value of the impure functions does not solely depend on its arguments passed.
• They do not modify the arguments which are passed to them.	• They may modify the arguments which are passed.
• Example: strlen(), sqrt()	• Example: random(), Date()

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

- Modifying the variable outside of function causes side effect.
- Example:

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

  - Here, the result of ***inc()*** will change every time if the value of ***'y'*** get changed inside the function definition.
  - Hence, the side effect of inc () function is changing the data of the external variable ***'y'.***

Section - D

Answer the following questions: **(5 Marks)**

1. What are called Parameters and write a note on

(i) Parameter without Type (ii) Parameter with Type

Answer:

- **Parameters** are the variables in a function definition
- **Arguments** are the values which are passed to a function definition.
- Two types of parameter passing are,
  1. Parameter Without Type
  2. Parameter With Type

1. Parameter Without Type:

- Lets see an example of a function definition of Parameter Without Type:

```
(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** passed **to** the variable ***'b'*** is the **argument**.
- The precondition ***(requires)*** and postcondition ***(returns)*** of the function is given.
- We have not mentioned any types: ***(data types)***. This is called parameter without type.

- In the above function definition the expression has type ***'int',*** so the function's return type also be ***'int'*** *by implicit.*

2. Parameter With Type:

- Now let us write the same function definition with types,

```
(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)
```

- In this example we have explicitly annotating the types of argument and return type as ***'int'***.
- Here, when we write the type annotations for 'a' and 'b' the parantheses are mandatory.
- This is the way passing parameter with type which helps the compiler to easily infer them.

2. 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

- ***gcd***

ii) Identify the statement which tells it is a recursive function

- ***let rec gcd a b :=***
- **"rec" keyword tells the compiler it is a recursive function**

iii) Name of the argument variable

- **'a'** and **'b'**

iv) Statement which invoke the function recursively

- ***gcd b (a mod b)***

v) Statement which terminates the recursion

- ***return a***

3. Explain with example Pure and impure functions.

Pure Function	Impure Function
• Pure functions will give exact result when the same arguments are passed.	• Impure functions never assure you that the function will behave the same every time it's called.
• Pure function does not cause any side effects to its output.	• Impure function causes side effects to its output.
• The return value of the pure functions solely depends on its arguments passed.	• The return value of the impure functions does not solely depend on its arguments passed.
• They do not modify the arguments which are passed to them	• They may modify the arguments which are passed.
• If we call pure functions with same set of arguments, we will always get the same return values.	• If we call pure functions with same set of arguments, we might get the different return values.
Example: sqrt() *let square x* *return: x * x*	• **Example: random()** *let Random number* *let a := random()* *if a > 10 then* *return: a* *else* *return: 10*

4. Explain with an example interface and implementation.

➢ **Interface**

- An interface is a set of action that an object can do.
- Interface just defines what an object can do, but won't actually do it.
- The interface defines an object's visibility to the outside world.
- In Object Oriented Programming language, an Interface is a description of all functions that a class must have.
- The purpose of interfaces is to allow the computer to enforce the properties of the class which means the class of ***TYPE T*** *(whatever the interface is)* must have functions called *X*, *Y*, *Z*, etc.

- For example when you press a light switch, the light goes on, you may not have cared how it splashed the light
- In our example, anything that ***"ACTS LIKE"*** a light, should have function definitions like turn_on () and a turn_off ().
- An object ***"ACTS LIKE"*** is an instance created from the class ***"LIGHT"***. All the objects of class ***"LIGHT"*** will uses all its functions.

➢ **Characteristics of interface:**

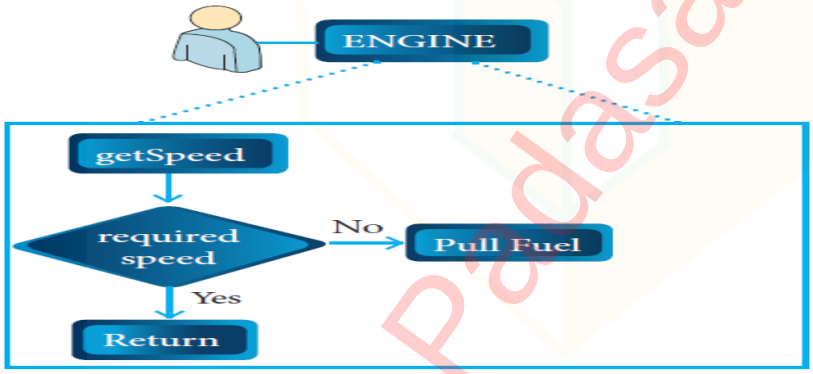
- The class template specifies the interfaces to enable an object to be created and operated properly.
- An object's attributes and behaviour is controlled by sending functions to the object.

➢ **Implementation:**

- Implementation carries out the instructions defined in the interface
- How the object is processed and executed is the implementation.
- A class declaration combines the external interface *(its local state)* with an implementation of that interface *(the code that carries out the behaviour)*.

➢ **Example:**

Let's take the example of increasing a car's speed.



- ➢ 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.

PREPARED BY

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com
Puducherry.

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com
Puducherry.

(06-07-2019)

COMPUTER SCIENCE

2. DATA ABSTRACTION

Section – A

Choose the best answer **(1 Mark)**

1. Which of the following functions that build the abstract data type ?

(A) Constructors (B) Destructors (C) recursive (D)Nested

2. Which of the following functions that retrieve information from the data type?

(A) Constructors **(B) Selectors** (C) recursive (D)Nested

3. The data structure which is a mutable ordered sequence of elements is called

(A) Built in **(B) List** (C) Tuple (D) Derived data

4. A sequence of immutable objects is called

(A) Built in (B) List **(C) Tuple** (D) Derived data

5. The data type whose representation is known are called

(A) Built in datatype (B) Derived datatype

(C) Concrete datatype (D) Abstract datatype

6. The data type whose representation is unknown are called

(A) Built in datatype (B) Derived datatype

(C) Concrete datatype **(D) Abstract datatype**

7. Which of the following is a compound structure?

(A) Pair (B) Triplet (C) single (D) quadrat

8. Bundling two values together into one can be considered as

(A) Pair (B) Triplet (C) single (D) quadrat

9. Which of the following allow to name the various parts of a multi-item object?

(A) Tuples (B) Lists **(C) Classes** (D) quadrats

10. Which of the following is constructed by placing expressions within square brackets?

(A) Tuples **(B) Lists** (C) Classes (D) quadrats

Section-B

Answer the following questions **(2 Marks)**

1. What is abstract data type?

- **Abstract Data type (ADT)** is a type or class for objects whose behavior is defined by a set of value and a set of operations.

2. Differentiate constructors and selectors.

CONSTRUCTORS	SELECTORS
• Constructors are functions that build the abstract data type.	• Selectors are functions that retrieve information from the data type.
• Constructors create an object, bundling together different pieces of information	• Selectors extract individual pieces of information from the object.

3. What is a Pair? Give an example.

- Any way of bundling two values together into one can be considered as a pair.
- Lists are a common method to do so.
- Therefore List can be called as Pairs.
- **Example:** lst[(0,10),(1,20)]

4. What is a List? Give an example.

- List can store multiple values of any type.
- List is constructed by placing expressions within square brackets separated by commas.
- Such an expression is called a list literal.
- **Example:** lst[10,20]

5. What is a Tuple? Give an example.

- A tuple is a comma-separated sequence of values surrounded with parentheses.
- Tuple is similar to a list.
- Cannot change the elements of a tuple.
- **Example:** Color= ('red', 'blue', 'Green')

Section-C

Answer the following questions **(3 Marks)**

1. Differentiate Concrete data type and abstract datatype.

CONCRETE DATA TYPE	ABSTRACT DATA TYPE
• Concrete data types or structures (CDT's) are direct implementations of a relatively simple concept.	• Abstract Data Types (ADT's) offer a high level view (and use) of a concept independent of its implementation.
• A concrete data type is a data type whose representation is known.	• Abstract data type the representation of a data type is unknown.

2. Which strategy is used for program designing? Define that Strategy.

- A powerful strategy for designing programs is '**wishful thinking**'.
- Wishful Thinking is the formation of beliefs and making decisions according to what might be pleasing to imagine instead of by appealing to reality.

3. Identify Which of the following are constructors and selectors?

(a) N1=number() -- **Constructor**

(b) accetnum(n1) -- **Selector**

(c) displaynum(n1) -- **Selector**

(d) eval(a/b) -- **Selector**

(e) x,y= makeslope (m), makeslope(n) -- **Constructor**

(f) display() -- **Selector**

4. What are the different ways to access the elements of a list. Give example.

- The elements of a list can be accessed in two ways.

1. Multiple Assignment:

- Which unpacks a list into its elements and binds each element to a different name.

Example:

```
lst := [10, 20]
x, y := lst
```

➢ ***x*** will become10 and ***y*** will become 20.

2. Element Selection Operator:

- It is expressed using square brackets.
- Unlike a list literal, a square-brackets expression directly following another expression does not evaluate to a list value, but instead selects an element from the value of the preceding expression.

Example:

```
lst[0]
10
lst[1]
20
```

5. Identify Which of the following are List, Tuple and class ?

(a) arr [1, 2, 34]	--	**List**
(b) arr (1, 2, 34)	--	**Tuple**
(c) student [rno, name, mark]	--	**Class**
(d) day= ('sun', 'mon', 'tue', 'wed')	--	**Tuple**
(e) x= [2, 5, 6.5, [5, 6], 8.2]	--	**List**
(f) employee [eno, ename, esal, eaddress]	--	**Class**

Section - D

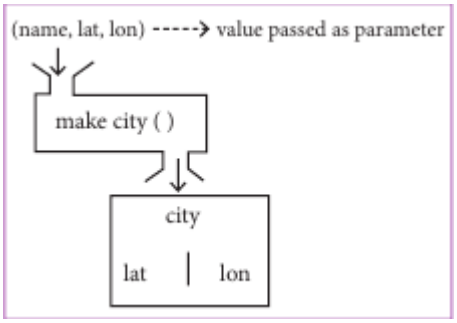
Answer the following questions: **(5 Marks)**

1. How will you facilitate data abstraction. Explain it with suitable example.

- Data abstraction is supported by defining an abstract data type (ADT), which is a collection of constructors and selectors.
- To facilitate data abstraction, you will need to create two types of functions:
  - **Constructors**
  - **Selectors**

a) Constructor:

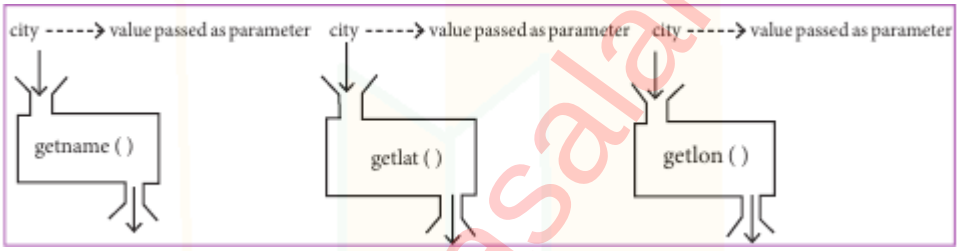
- Constructors are functions that build the abstract data type.
- Constructors create an object, bundling together different pieces of information.
- For example, say you have an abstract data type called city.
- This city object will hold the city's name, and its latitude and longitude.
- To create a city object, you'd use a function like **city = makecity (name, lat, lon).**
- Here makecity (name, lat, lon) is the constructor which creates the object city.



b) Selectors:

- Selectors are functions that retrieve information from the data type.
- Selectors extract individual pieces of information from the object.
- To extract the information of a city object, you would use functions like
  - **getname(city)**
  - **getlat(city)**
  - **getlon(city)**

  These are the selectors because these functions extract the information of the city object.



2. What is a List? Why List can be called as Pairs. Explain with suitable example.

LIST:

- List is constructed by placing expressions within square brackets separated by commas.
- Such an expression is called a list literal.
- List can store multiple values.
- Each value can be of any type and can even be another list.
- The elements of a list can be accessed in two ways.

1. Multiple Assignment:

- Which unpacks a list into its elements and binds each element to a different name.

Example:

```
lst := [10, 20]
x, y := lst
```

- ***x*** will become10 and ***y*** will become 20.

2. Element Selection Operator:

- It is expressed using square brackets.
- Unlike a list literal, a square-brackets expression directly following another expression does not evaluate to a list value, but instead selects an element from the value of the preceding expression.

Example:

lst[0]
10
lst[1]
20

PAIR:

- Any way of bundling two values together into one can be considered as a pair.
- Lists are a common method to do so.
- Therefore List can be called as Pairs.

Example: lst[(0,10),(1,20)]

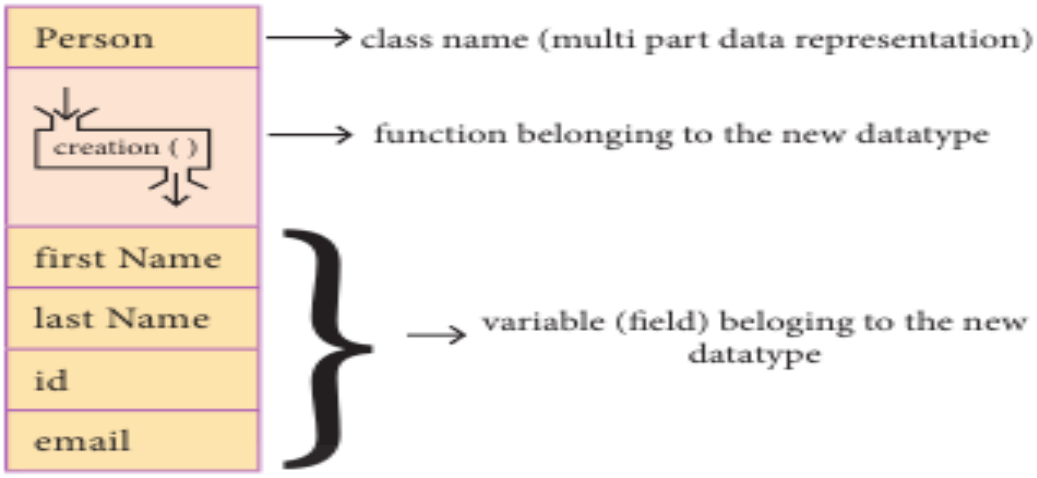
3. How will you access the multi-item. Explain with example.

MULTI-ITEM:

- The structure construct in OOP languages it's called **class construct** is used to represent multi-part objects where each part is named.
- Consider the following pseudo code:

```
class Person:
    creation( )
    firstName := " "
    lastName := " "
    id := " "
    email := " "
```

- The new data type Person is pictorially represented as,



Let main() contains	
p1:=Person()	statement creates the object.
firstName := " Padmashri "	setting a field called firstName with value Padmashri
lastName :="Baskar"	setting a field called lastName with value Baskar
id :="994-222-1234"	setting a field called id value 994-222-1234
email="compsci@gmail.com"	setting a field called email with value compsci@gmail.com
- - output of firstName : Padmashri	

- The class (structure) construct defines the form for multi-part objects that represent a person.
- Person is referred to as a class or a type, while p1 is referred to as an object or an instance.
- Using class you can create many objects of that type.
- Class defines a data abstraction by grouping related data items.
- A class as bundled data and the functions that work on that data that is using class we can access multi-part items.

PREPARED BY

J. BASKARAN M.Sc., B.Ed. (C.S)
jbaskaran89@gmail.com
Puducherry.

J. ILAKKIA M.Sc., M.Phil., B.Ed. (C.S)
jilakkia@gmail.com
Puducherry.