

12th



2, 3 &
5 Mark

COMPUTER SCIENCE

Name:.....

Class:.....Sec.....

Roll No:.....

School:.....

CHAPTER	CONTENTS	P.NO
1	FUNCTION	
2	DATA ABSTRACTION	
3	SCOPING	
4	ALGORITHMIC STRATEGIES	
5	PYTHON- VARIABLES AND OPERATORS	
6	CONTROL STRUCTURES	
7	PYTHON FUNCTIONS	
8	STRINGS AND STRING MANIPULATION	
9	LISTS, TUPLES, SETS & DICTIONARY	
10	PYTHON CLASSES & OBJECTS	
11	DATA BASE CONCEPTS	
12	STRUCTURAL QUERY LANGUAGE	
13	PYTHON AND CSV FILES	
14	IMPORTING C++ PROGRAMS IN PYTHON	
15	DATA MANIPULATION THROUGH SQL	
16	DATA VISUALIZATION USING PYPLOT:LINE CHART, PIE & BAR	

CHAPTER	1 M		2M		3M		5M		Total	Sign
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										

**Mr. D.JENIS M.C.A., B.Ed.,
Muhyiddeen Matric. Hr. Sec. School,
Kayalpatnam-Tuticorin Dt.,
Email: jenisroyal@gmail.com**

1. FUNCTION

2 Mark

1. What is a 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 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 sum x y:

return x+y

Ans: Normal 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

3 Mark

1. Mention the characteristics of Interface.

The **class template** specifies the interface to enable an object to be created and operated properly.

An **objects'** attribute and behavior is controlled by sending functions to the object.

2. Why strlen is called pure function?

```

s="Name"
let length s:=
  i:=0
  if i < strlen(s) then
    print s[i]
    ++i
Output
4

```

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.

side effect of impure function is that it **doesn't take any arguments** and it **doesn't return any value**. It is depends on variables or functions **outside of its definition block**. If you call the impure functions with the same set of arguments, you might get the **different return values**.

Example

```

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

```

→ inc(4) // y=4
 → inc(4) // y=8
 → inc() will **change every time** if the value of 'y' get changed inside the function definition

4. Differentiate pure and impure function

Pure Function	Impure Function
Pure functions will give exact result when the same arguments are passed.	Impure functions with the same set of arguments, you might get the different return values.
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
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 the function **causes side effect**.

Example:

```

let y:=0

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

```

→ inc(4) // y=4
→ inc(4) // y=8
→ inc( ) will change
every time if the value
of 'y' get changed
inside the function
definition
  
```

5 Mark

1. What are called Parameters and write a note on

(i) Parameter without Type

ii) Parameter with Type

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

Parameter Without Type

(requires : $b \geq 0$)

(returns : a to the power of b)

let rec pow a b:=

if b=0 then 1

*else a * pow b(a-1)*

The **precondition** (requires) and **post condition** (returns) of the function is given. We have **not mentioned any data types**. This is called **parameter without type**. The **expression** has type 'int', so the functions **return type** also be 'int' by implicit.

Parameter With Type

(requires : $b \geq 0$)

(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)*

The types of **argument** and **return type** as 'int'. The **type annotations** for 'a' and 'b' the **parentheses** 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

Ans: gcd

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

Ans: let rec gcd a b :=

“rec” keyword tells the compiler it is a recursive function.

iii) Name of the argument variable

Ans: ‘a’ and ‘b’

iv) Statement which invoke the function recursively

Ans: gcd b(a mod b)

v) Statement which terminates the recursion

Ans: 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 with the same set of arguments, you might get the different return values.
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: <pre> let square x return: x * x </pre>	Example: <pre> let a:= random() let random() := if a>10 then return: a else return: 10 </pre>

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 **class template** specifies the interface to enable an object to be created and operated properly.
- An **objects'** attribute and behavior is controlled by sending functions to the object.

Example

- ❑ when you **press a light switch**, the light goes on, you may not have cared how is splashed the light .
- ❑ A light should have function definitions like **turn_on()** and **turn_off()**.

Implementation

- Implementation carries out the **instructions** defined in the interface.
- A class declaration combines the **external interface** with an **implementation** of the interface.
- An **object** is an instance created from the class

Example

- ❑ 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 behavior. This is the interface.
- ❑ Internally, the **engine of the car** is doing all things.
- ❑ Its' where **fuel, air, pressure, and electricity** come together to create the power to move the vehicle.
- ❑ Thus we **separate** interface from **implementation**.

2. DATA ABSTRACTION

2 Mark

1. What is abstract data type?

Abstract Data type 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 is constructed by **placing expressions** within **square brackets** **separated by commas**. Such an **expression** is called a **list literal**. List can store multiple values.

Example

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

```
x, y := lst // x will become 10 and y will become 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

```
colour = ('red', 'blue', 'green')
```

3 Mark

1. Differentiate Concrete data type and abstract data type.

Concrete Data Type	Abstract Data Type
Concrete data types or structures are direct implementations of a relatively simple concept.	Abstract Data types offer a high level view of a concept independent of its implementation.
Concrete Data Type is a data type whose representation is known .	Abstract Data Type is 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: **'wishful thinking'**.
- Wishful Thinking is the formation of **beliefs and making decisions**.
- According to what might be **pleasing to imaging** instead of by **appealing to reality**.

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

a	N1= number()	a	Constructors
b	accetnum(n1)	b	Selectors
c	displaynum(n1)	c	Selectors
d	eval(a/b)	d	Selectors
e	x, y = makeslope (m), makeslope(n)	e	Constructors
f	display ()	f	Selectors

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

The **first** way method of *multiple assignments*, which unpacks a list into its elements and binds **each element to a different name**.

Example

```
lst := [10,20]
x, y := lst // x will become 10 and y will become 20
```

The **second** method for accessing the elements in a list is by the *element selection operator*, also expressed using square brackets.

Example

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

5. Identify which of the following are List, Tuple and Class?

a	arr[1,2,3,4]	a	List
b	arr(1,2,34)	b	Tuple
c	student[rno, name, mark]	c	Class
d	day = ('sun', 'mon', 'tue', 'wed')	d	Tuple
e	x = [2, 5, 6.5,[5,6],8.2]	e	List
f	employee[eno, ename, esal, eaddress]	f	Class

5 Mark

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

Data abstraction is supported by **defining an abstract data type**, which is a collection of **constructors and selectors**.

Constructors

- Constructors are **functions** that **build the abstract data type**.
- Constructors **create an object**, **bundling together different pieces of information**.

Example

```
city = makecity(name, lat, lon)
```

Here **makecity(name, lat, lon)** is the **constructor** which creates the **object city**.

Selectors

- Selectors are **functions** that **retrieve information from the data type**.
- Selectors extract individual pieces of information from the object.

Example

```
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**.

The first way method of *multiple assignments*, which unpacks a list into its elements and binds **each element to a different name**.

Example

```
lst := [10,20]
x, y := lst // x will become 10 and y will become 20
```

The second method for **accessing the elements in a list** is by the **element selection operator**, also expressed using **square brackets**.

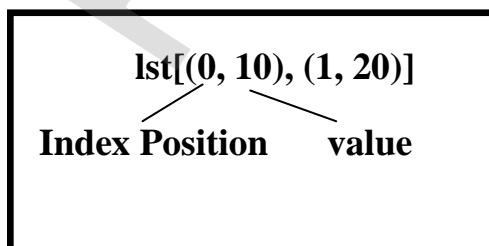
Example

```
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



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

The **class** or **structure** construct to represent **multi-part of objects** where each part is **named**.

Pseudo code

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

Person	Class name (multipart data representation)
creation()	function belonging to the new data type
first Name	variable belonging to the new data type
last Name	
id	
email	

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

The **class** construct defines the form for **multi-part objects** that represent **person**. **Person** is referred to as a **class or type**, while p1 is referred to as an **object** or an **instance**. A **class** as **bundled data** and the **functions** that **work on that data**.