# UNIT-3

1 MARKS Q&A

## a)Write any two benefits of using functions in Python.

A: **Two benefits of using functions in Python:**

**Code Reusability:** Functions allow you to reuse code, reducing duplication and making the program easier to maintain.

**Modularity:** They break down complex problems into smaller, manageable pieces, improving code organization and readability.

## b) List the various meta characters available in re module.

A:  . – Matches any character except a newline.

 ^ – Matches the start of the string.

 $ – Matches the end of the string.

 * – Matches 0 or more repetitions of the preceding character.

 + – Matches 1 or more repetitions of the preceding character.

 ? – Matches 0 or 1 occurrence of the preceding character.

## c) Define recursion in Python.

A: **Recursion** is a process where a function calls itself to solve a problem. It continues until a base condition is met to terminate the recursion.

## d) State the purpose of math module in python.

A: The math module provides various mathematical functions and constants, such as:

- math.sqrt() – Computes the square root.
- math.pow() – Raises a number to a power.
- math.pi – Returns the value of π.

## e) List the special symbols used in re module.

A:  \A – Matches the beginning of a string.

 \Z – Matches the end of a string.

 \d – Matches any digit (0-9).

 \D – Matches any non-digit.

 \s – Matches any whitespace.

 \S – Matches any non-whitespace.

**3 MARKS Q&A**

## a)Define function in Python. Mention the types of function and state its uses.

**A: Definition:**

A function in Python is a block of code that performs a specific task and can be reused multiple times. It is defined using the def keyword.

🔥 **Types of Functions:**

1. Built-in Functions: Predefined functions such as print(), len(), type(), etc.
2. User-defined Functions: Functions created by the user to perform specific operations.
3. Anonymous (Lambda) Functions: Single-line, unnamed functions using the lambda keyword.
4. Recursive Functions: Functions that call themselves to solve a problem.

🎯 **Uses of Functions:**

- Code Reusability: Reduces duplication by reusing code.
- Modularity: Breaks down a large program into smaller, manageable parts.
- Improved Readability: Makes code more organized and easier to maintain.

## b)Write a python program using function to find the factorial of a given number using recursion.

**A:Program:**

```
# Function to calculate factorial using recursion
def factorial(n):
    if n == 1 or n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Input from user
num = int(input("Enter a number: "))
```

```
# Check for negative number
if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    print(f"The factorial of {num} is {factorial(num)}")
```

**c)Write a python program using function to find the sum of first 'n' numbers.**

**A:Program:**

```
# Function to calculate sum of first n numbers
def sum_of_n(n):
    return n * (n + 1) // 2

# Input from user
n = int(input("Enter a number: "))

# Check for negative input
if n < 0:
    print("Sum cannot be calculated for negative numbers.")
else:
    print(f"The sum of first {n} numbers is {sum_of_n(n)}")
```

**d) Write short notes on local variables and global variables.**

**A: 🌐 1. Global Variables:**

- Defined outside any function and accessible throughout the program.
- Can be accessed and modified within a function using the global keyword.

**EXAMPLE**:

```
x = 10  # Global variable

def my_function():
    global x
    x = 20  # Modifying global variable
    print("Inside function:", x)

my_function()
print("Outside function:", x)
```

🔒 **2. Local Variables:**

- Defined **inside** a function and accessible only within that function.
- They cannot be accessed outside the function.

**EXAMPLE:**

```
def my_function():
    y = 5  # Local variable
    print("Inside function:", y)

my_function()
# print(y)  # This will raise an error as y is not accessible here.
```

## e) Explain the purpose of re module in python.

**A: 📚 Purpose of re Module:**

The re module in Python is used for pattern matching and text processing using regular expressions. It allows you to search, match, and manipulate strings.

🔥 **Common Functions in re Module:**

1. re.match(pattern, string):
   - Checks if the pattern matches only at the beginning of the string.
   - Returns a match object if successful, otherwise None.
2. re.search(pattern, string):
   - Searches for the first occurrence of the pattern anywhere in the string.
   - Returns a match object if successful, otherwise None.
3. re.findall(pattern, string):
   - Returns all non-overlapping matches of the pattern in a list.

## 5 MARKS Q&A

### a) Explain in detail about the random module in Python with a suitable example.

**A:**The random module in Python provides a variety of functions to generate random numbers, shuffle data, and choose random elements. It is commonly used in simulations, games, security, and testing.

| Function | Description |
|---|---|
| randint(a, b) | Random integer between a and b |
| random() | Random float between 0.0 and 1.0 |
| choice(seq) | Random item from a sequence |
| shuffle(seq) | Shuffles a list |
| uniform(a, b) | Random float between a and b |

Ex:

```
import random
import string

# a) Generate a random integer between 1 and 10
num = random.randint(1, 10)
print(f"Random number between 1 and 10: {num}")

# b) Generate a random float between 0 and 1
num_float = random.random()
print(f"Random float between 0 and 1: {num_float}")

# c) Choose a random item from a list
colors = ['red', 'blue', 'green', 'yellow']
choice = random.choice(colors)
print(f"Randomly selected color: {choice}")

# d) Shuffle a list of numbers
numbers = [1, 2, 3, 4, 5]
random.shuffle(numbers)
print(f"Shuffled list: {numbers}")

# e) Generate a random float between 1 and 100
num_uniform = random.uniform(1, 100)
print(f"Random float between 1 and 100: {num_uniform}")

# f) Generate a 6-character random password
```

```
password = ''.join(random.choices(string.ascii_letters + string.digits, k=6))
print(f"Generated Password: {password}")
```

## b) Describe how to declare and call functions in Python programs. Illustrate with an example script.

**A: 📚 1. Declaring a Function:**

- Use the def keyword followed by the function name and parentheses () containing optional parameters.
- Define the function body with indented code.
- Use the return statement to return a result (optional).

Python.

**🔥 2. Calling a Function:**

- Call the function by using its name followed by parentheses () and pass required arguments.
- Store the result in a variable or directly print it.

EXAMPLE:

```
# Function to calculate the square of a number
def square(num):
    """Returns the square of the given number."""
    return num * num

# Calling the function
n = 5
result = square(n)
print(f"The square of {n} is {result}")
```

## c)Write short notes on match( ), search( ), and findall( ) functions in Python with relevant examples.

**A: 📚 1. re.match()**

- Checks for a match only at the beginning of the string.
- Returns a match object if successful, otherwise None.

Python.

Ex:

```
import re
```

```
text = "Hello World"
result = re.match(r'Hello', text)

if result:
    print("Match found:", result.group())  # Output: Hello
else:
    print("No match found")
```

🔍 **2. re.search()**

- Searches for the first occurrence of the pattern anywhere in the string.
- Returns a match object if successful, otherwise None.

Python.

Ex:

```
    result = re.search(r'World', text)
if result:
   print("Pattern found:", result.group())  # Output: World
else:
   print("Pattern not found")
```

📄 **3. re.findall()**

- Returns all occurrences of the pattern in the string as a list.

Python.

Ex:

```
    text2 = "The year is 2025 and the month is 03"
result = re.findall(r'\d+', text2)
print("All matches:", result)  # Output: ['2025', '03']
```

**d)Develop a python program to check whether a given number is in PAN format or not.**

**A:Program:**

```
import re

def is_valid_pan(pan):
    pattern = r'^[A-Z]{5}\d{4}[A-Z]$'

    # Check if PAN matches the pattern
    if re.match(pattern, pan):
        return "Valid PAN Number"
    else:
        return "Invalid PAN Number"

# Input from user
pan_number = input("Enter PAN number: ")
print(is_valid_pan(pan_number))
```

## e)Explain in detail about meta characters in re module with a suitable example.

**A: 📚 What are Meta Characters?**

Meta characters in regular expressions are special characters that control how patterns are matched.

### 🔥 Common Meta Characters:

1. **.** – Matches any character except a newline.

Python

Ex:

```
import re
result = re.findall(r'H.llo', 'Hello, Hallo, Hillo')
print(result)  # Output: ['Hello', 'Hallo', 'Hillo']
```

2. ^ – Matches the start of a string.

Python.

Ex:

```
result = re.match(r'^Hello', 'Hello World')
print(result.group())  # Output: Hello
```

3. **$** – Matches the end of a string.

Python.

Ex:

```
result = re.search(r'World$', 'Hello World')
print(result.group())  # Output: World
```

4. ***** – Matches 0 or more occurrences of the preceding character.

Python.

Ex:
```
result = re.findall(r'ab*', 'ab, abb, abbb, a')
print(result)  # Output: ['ab', 'abb', 'abbb', 'a']
```

5. **+** – Matches 1 or more occurrences of the preceding character.
Python.

Ex:
```
result = re.findall(r'ab+', 'ab, abb, abbb, a')
print(result)  # Output: ['ab', 'abb', 'abbb']
```

## 10 MARKS Q&A

### a) Examine how python supports regular expressions through the 're' module with a brief introduction and the explanation of various special symbols related to it.

### A: 📚 Introduction to re Module:

- The re module in Python provides support for regular expressions (regex).
- Regular expressions are used to search, match, and manipulate strings based on patterns.

### 🔥 Commonly Used Functions in re Module:

1. re.match() – Matches a pattern only at the beginning of the string.
2. re.search() – Searches for a pattern anywhere in the string.
3. re.findall() – Returns a list of all matching patterns.
4. re.sub() – Replaces the pattern with another string.

## 🎯 Special Symbols in `re` Module:

| Symbol | Description | Example |
|---|---|---|
| `.` | Matches any single character | `a.b` → matches `acb` |
| `^` | Matches the start of a string | `^Hello` → matches `Hello World` |
| `$` | Matches the end of a string | `world$` → matches `Hello world` |
| `*` | Matches 0 or more repetitions | `a*` → matches `aaa` or empty |
| `+` | Matches 1 or more repetitions | `a+` → matches `a`, `aa` |
| `?` | Matches 0 or 1 occurrence | `colou?r` → matches `color` or `colour` |
| `{n}` | Matches exactly `n` repetitions | `a{3}` → matches `aaa` |
| `[]` | Matches any character inside brackets | `[abc]` → matches `a`, `b`, or `c` |
| `\d` | Matches a digit (0–9) | `\d+` → matches `123` |
| `\w` | Matches a word character | `\w+` → matches `abc123` |

**Ex:**

```
    import re

text = "Hello, my phone number is 9876543210 and my email is
abc@gmail.com"

# Match a phone number pattern
phone_pattern = re.search(r'\d{10}', text)
if phone_pattern:
   print(f"Phone number found: {phone_pattern.group()}")

# Find all words
words = re.findall(r'\w+', text)
print(f"Words found: {words}")
```

**b) Describe what a Module is in Python. Explain how you can use math module in python with an example.**

**A: 📚 What is a Module?**

- A module is a Python file that contains functions, classes, and variables that can be imported and reused in other programs.
- Modules help organize code and avoid repetition.

**🎯 Types of Modules:**

1. **Built-in Modules:** Pre-installed modules (e.g., math, random, os).
2. **User-defined Modules:** Created by the user.

### 🔢 Using math Module in Python

- The math module provides useful mathematical functions.

### ✅ Importing the math Module:

```
import math
```

### 📝 Commonly Used Functions in math Module:

| Function | Description | Example |
|---|---|---|
| math.sqrt(x) | Returns the square root of x | math.sqrt(16) → 4.0 |
| math.factorial(x) | Returns the factorial of x | math.factorial(5) → 120 |
| math.ceil(x) | Rounds x to the next integer | math.ceil(4.3) → 5 |
| math.floor(x) | Rounds x to the previous integer | math.floor(4.7) → 4 |
| math.pi | Returns the value of Pi | math.pi → 3.14159 |

Ex:

```
import math

# Calculate square root
num = 25
print(f"Square root of {num}: {math.sqrt(num)}")

# Calculate factorial
print(f"Factorial of 5: {math.factorial(5)}")

# Round up a number
print(f"Ceiling of 4.3: {math.ceil(4.3)}")

# Round down a number
print(f"Floor of 4.7: {math.floor(4.7)}")
```

**c) Describe argument in python functions and explain in detail how to pass parameters to functions with a suitable example.**

**A: 📚 What is an Argument in Python?**

- An argument is a value passed to a function when it is called.
- Functions can accept multiple arguments and return results.

## 🎯 Types of Arguments in Python:

1. Positional Arguments: Passed in the order in which they appear.
2. Keyword Arguments: Passed using key=value format.
3. Default Arguments: Arguments with default values.
4. Variable-length Arguments: Allows passing multiple values.

**Python supports several types of arguments:**

1. Positional Arguments:
- Passed to a function in the order they are defined.
- The function matches arguments to parameters based on their position.

2. Keyword Arguments:
- Passed to a function with a keyword (parameter name) and a value.
- Allows arguments to be passed out of order.

3. Default Arguments:
- Assigned a default value in the function definition.
- If a value is not provided when the function is called, the default value is used.

4. Variable-Length Arguments (*args):
- Allows a function to accept an arbitrary number of positional arguments.
- These arguments are packed into a tuple.

5. Variable-Length Keyword Arguments (**kwargs):
- Allows a function to accept an arbitrary number of keyword arguments.
- These arguments are packed into a dictionary.

**Ex:**

```
def greet(name, greeting="Hello"):
    """Greets a person with an optional greeting."""
    print(f"{greeting}, {name}!")

# Example 1: Using both positional and default arguments
greet("Alice")  # Output: Hello, Alice!

# Example 2: Overriding the default argument
greet("Bob", "Hi")  # Output: Hi, Bob!
```

```
# Example 3: Using keyword arguments
greet(name="Charlie", greeting="Good morning") #Output: Good
morning, Charlie!

def add_numbers(*numbers):
    """Adds any number of numbers."""
    total = 0
    for number in numbers:
        total += number
    return total

result = add_numbers(1, 2, 3, 4)
print(result)  #Output: 10

def person_info(**info):
    """Prints person info"""
    for key, value in info.items():
        print(f"{key}: {value}")

person_info(first_name="David", last_name="Lee", age=25)
#Output:
#first_name: David
#last_name: Lee
#age: 25
```