RTPNOTES.vercel.app

Python-Module-1-Important-Topics

For more notes visit

https://rtpnotes.vercel.app

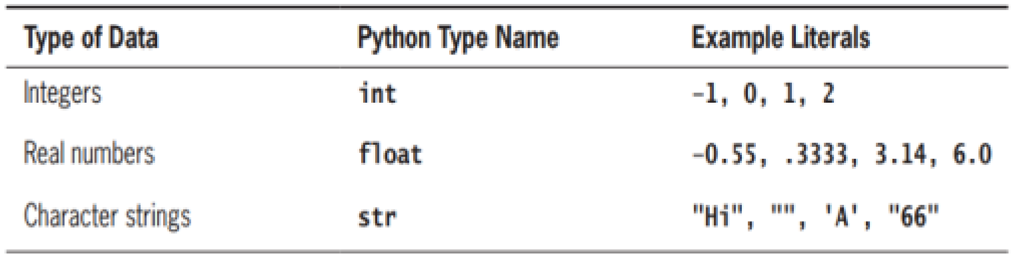
RTPNOTES.vercel.app

  - 8. Write a Python program to reverse a number and also find the sum of digits of the number.Prompt the user for input
  - 9. Explain the concept of scope and lifetime of variables in Python programming language, with a suitable example.
    - Example
- Python-Module-1-University-Questions-Part-B
  - 1. Write a python program to generate the following type of pattern for the given N rows .
  - 2. Write the python program to print all prime numbers less than 1000.
  - 3. Write a Python program to find distance between two points (x1,y1) and (x2,y2).
  - 4. Write a Python program to find the roots of a quadratic equation, ax2+ bx + c =0 .
  - 5. Write a Python program to check whether a number is Armstrong number or not.
  - 6. Write a Python program to display the sum of odd numbers between a programmer specified upper and lower limit.
  - 7. Given the value of x, write a Python program to evaluate the following series upto n terms:
  - 8. Write a python program to find X^Y or pow(X,Y) without using standard function.
  - 9. Write a python program to generate the following type of pattern for the given N rows where N <= 26.
  - 10. Discuss the steps involved in the waterfall model of software development process with the help of a neat diagram.
  - 11. Write a Python program to print all numbers between 100 and 1000 whose sum of digits is divisible by 9.
  - 12. Illustrate the use of range() in Python.
    - What is range?
    - Example
  - 13. Write a Python program to print all prime factors of a given number.
  - 14. Write a Python code to check whether a given year is a leap year or not
  - 15. What are the possible errors in a Python program.
  - 16. Write a Python program to print the value of 2^(2n)+n+5 for n provided by the user.
  - 17. Write a Python code to determine whether the given string is a Palindrome or not using slicing. Do not use any string function.

RTPNOTES.vercel.app

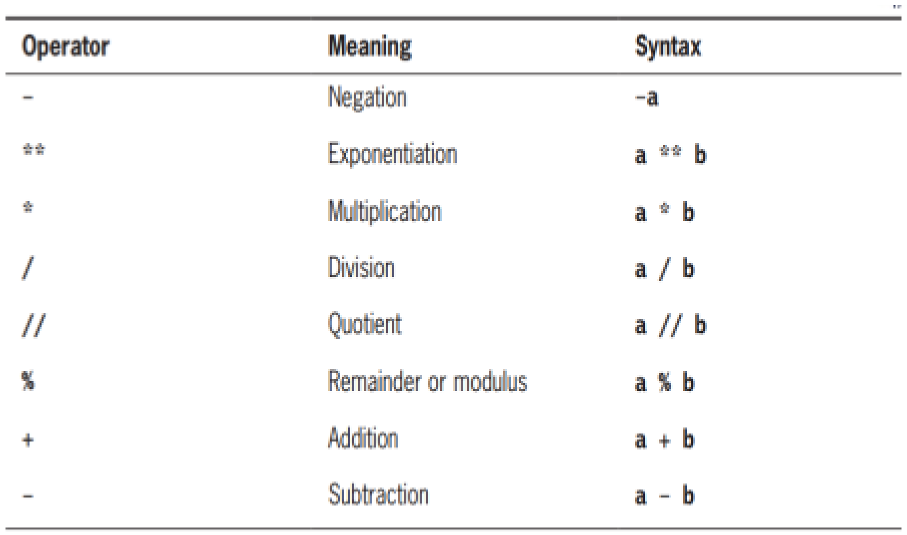
1. Data types

- A data type consists of a set of values and a set of operations that can be performed on those values.
- A literal is the way a value of a data type looks to a programmer. The programmer can use a literal in a program to mention a data value.



Type of Data	Python Type Name	Example Literals
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', "66"`

2. Operator's



Operator	Meaning	Syntax
-	Negation	`-a`
**	Exponentiation	`a ** b`
*	Multiplication	`a * b`
/	Division	`a / b`
//	Quotient	`a // b`
%	Remainder or modulus	`a % b`
+	Addition	`a + b`
-	Subtraction	`a - b`

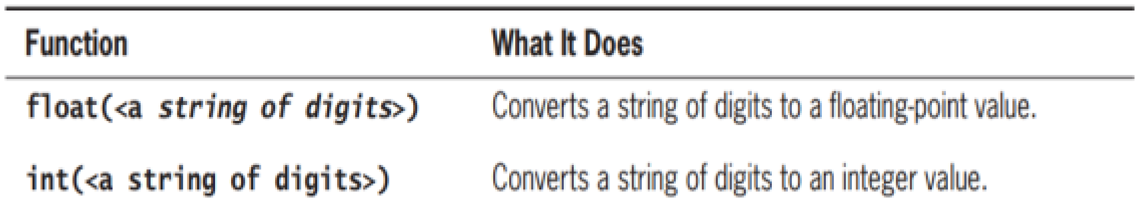
3. Expressions

Expressions provide an easy way to perform operations on data values to produce other data values.

Arithmetic Expressions
An arithmetic expression consists of operands and operators.

RTPNOTES.vercel.app

4. Int, float

- Integers - the integers include 0, all of the positive whole numbers, and all of the negative whole numbers.
- Floating - Point Numbers - A real number in mathematics, such as the value of pi (3.1416...), consists of a whole number, a decimal point, and a fractional part.
- 

Function	What It Does
`float(<a string of digits>)`	Converts a string of digits to a floating-point value.
`int(<a string of digits>)`	Converts a string of digits to an integer value.

5. Strings

- In python, a string literal is a sequence of characters enclosed in single or double quotation marks
- `"Hello There"`

6. Control statements

What are control statements

- Statements that allow the computer to select an action to perform in a particular action or to repeat a set of actions.
- Two types of control statements are there:
  - Selection structure
  - Iteration structure

Iteration Structure

- Also known as loops. Which repeat an action
- Each repetition of the action is known as an iteration
- There is for loops and while loops for doing this

RTPNOTES.vercel.app

For loop

Syntax

```
for <variable> in range(<an integer expression>):
        <statement-1>
        <statement-n>
```

Example

```
for eachPass in range(4):
        print("Its alive!")
```

Output

```
Its alive!
Its alive!
Its alive!
Its alive!
```

While loop

- Conditional iteration requires that a condition be tested within the loop to determine whether the loop should continue. Such a condition is called the loop's continuation condition.
- If the continuation condition is false, the loop ends. If the continuation condition is true, the statements within the loop are executed again

Syntax

```
while <condition>:
        <sequence of statements>
```

Example

```
num = 10
while(num>0):
```

RTPNOTES.vercel.app

```
        num = num - 1
        print("Subtracting")
```

Selection statements

- In some cases, instead of moving straight ahead to execute the next instruction, the computer might be faced with two alternative courses of action.
- The condition in a selection statement often takes the form of a comparison. The result of the comparison is a Boolean value True or False.

```
if <condition>:
        <sequence of statements-1>
elif <condition-n>
        <sequence of statements-n>
else:
        <default sequence of statements>
```

Example

```
num = 10
if(num > 10):
        print("Num greater than 10")
elif(num == 10):
        print("Num is equal to 10")
else:
        print("Num is less than 10")
```

Output

- Here Num is equal to 10 will be printed

7. Switch

- A switch statement is a very useful and powerful programming feature. It is an alternate to if-else-if ladder statement and provides better performance and more manageable code than an if-else-if ladder statement.
- Python does not provide inbuilt switch statements, we can implement on our own

RTPNOTES.vercel.app

Example

```
def get_week_day(argument):
switcher =
{
0: "Sunday" ,
1: "Monday" ,
2: "Tuesday" ,
3: "Wednesday" ,
4: "Thursday" ,
5: "Friday" ,
6: "Saturday"
}
return switcher.get(argument, "Invalid day")
print (get_week_day(6))
print (get_week_day(8))
print (get_week_day(0))
```

Output

```
Saturday
Invalid day
Sunday
```

Python-Module-1-University-Questions-Part-A

1. What is the output of the following print statement in Python? (a) print (9//2) (b) print (9/2)

(a) **4** (b) **4.5**

2. Write a Python program to count number of even numbers and odd numbers in a given set of n numbers.

RTPNOTES.vercel.app

```
def count_even_odd(numbers):
  """Counts the number of even and odd numbers in a list.

  Args:
      numbers: A list of integers.

  Returns:
      A tuple containing the count of even numbers and odd numbers.
  """

  even_count = 0
  odd_count = 0
  for num in numbers:
    if num % 2 == 0:
      even_count += 1
    else:
      odd_count += 1
  return even_count, odd_count

# Get the set of numbers from the user
n = int(input("Enter the number of elements: "))
numbers = []
for i in range(n):
  num = int(input("Enter element {}: ".format(i+1)))
  numbers.append(num)

# Count even and odd numbers
even_count, odd_count = count_even_odd(numbers)

# Print the results
print("Even numbers:", even_count)
print("Odd numbers:", odd_count)
```

3. What is the output of the following Python code. Justify your answer.

RTPNOTES.vercel.app

x = 'abcd'

for i in range(len(x)):

print(i)

```
0
1
2
3
```

- `range(len(x))` creates a sequence of numbers from 0 to `len(x)-1` (which is 3 in this case). So, the loop will iterate four times.
- so, 0,1,2,3 will be printed, as the range is [0,1,2,3]

4. Jack says that he will not bother with analysis and design but proceed directly to coding his programs. Why is that not a good idea?

- There are several reasons why skipping analysis and design and going straight to coding is not a good idea for Jack:
  - **Errors and rework:** Without proper analysis, Jack might miss crucial requirements or misunderstand the problem he's trying to solve. This can lead to errors in the code that may not be apparent until later stages, requiring significant rework.
  - **Inefficient solutions:** Without design, Jack might write code that works but is inefficient, poorly structured, or difficult to maintain. Design helps create a well-organized and optimized solution.
  - **Maintenance challenges:** Code written without proper planning can be difficult to understand and maintain for Jack himself or anyone else who might need to work on it later. This can lead to time-consuming debugging and modifications in the future.

5. Write the output of the following python statements :

i) round(12.57) ii) 5//2 iii) int(6.5)

RTPNOTES.vercel.app

- **round(12.57):** The `round` function rounds a number to the nearest integer. In this case, 12.57 is closer to 13 than 12, so the output is 13.
- **5//2:** The `//` operator performs integer division, which discards the remainder. Dividing 5 by 2 results in 2.5, but since we're using integer division, only the whole number part (2) is kept.
- **int(6.5):** The `int` function converts a number to an integer. In this case, 6.5 is a floating-point number. When converted to an integer, the decimal part is truncated, resulting in 6.

6. Explain type conversion with example.

```
num = 3.8
converted_num = int(num)
print(converted_num)  # Output: 3 (Decimal part truncated)
```

```
num = 10
converted_num = float(num)
print(converted_num)  # Output: 10.0 (Integer converted to float)
```

```
num = 123 converted_num = str(num) print(converted_num) # Output: "123"
(Integer converted to string)
```

7. Write a program that accepts the lengths of three sides of a triangle as inputs and outputs whether or not the triangle is a right triangle

```
def is_right_triangle(a, b, c):
  """
  Checks if the given side lengths form a right triangle.

  Args:
      a: Length of the first side.
      b: Length of the second side.
      c: Length of the third side.
```

RTPNOTES.vercel.app

```
    Returns:
        True if the triangle is right-angled, False otherwise.
    """

    # Sort the sides in ascending order (hypotenuse is always the longest)
    sides = sorted([a, b, c])
    hypotenuse = sides[-1]
    other_sides = sides[:-1]

    # Check if the Pythagorean theorem holds for the sorted sides
    return hypotenuse**2 == other_sides[0]**2 + other_sides[1]**2

# Get input from the user
while True:
    try:
        a = float(input("Enter the length of side a: "))
        b = float(input("Enter the length of side b: "))
        c = float(input("Enter the length of side c: "))

        # Validate that all sides are positive
        if a <= 0 or b <= 0 or c <= 0:
            raise ValueError("All sides of a triangle must be positive numbers.")

        break  # Exit the loop if valid input is entered
    except ValueError as e:
        print(e)
        print("Please enter valid positive numbers for the sides.")

# Check if it's a right triangle
if is_right_triangle(a, b, c):
    print("The triangle is a right triangle.")
else:
    print("The triangle is not a right triangle.")
```

8. Write a Python program to reverse a number and also find the sum of digits of the number.Prompt the user for input

RTPNOTES.vercel.app

```python
def reverse_and_sum_digits(num):
  reversed_num = 0
  sum_of_digits = 0
  while num > 0:
    digit = num % 10  # Extract the last digit
    reversed_num = reversed_num * 10 + digit  # Build the reversed number
    sum_of_digits += digit
    num //= 10  # Remove the last digit from the original number

  return reversed_num, sum_of_digits

# Get user input
number = int(input("Enter a positive integer: "))

# Reverse the number and find the sum of digits
reversed_number, sum_digits = reverse_and_sum_digits(number)

# Print the results
print("Original number:", number)
print("Reversed number:", reversed_number)
print("Sum of digits:", sum_digits)
```

9. Explain the concept of scope and lifetime of variables in Python programming language, with a suitable example.

Scope:

- Scope refers to the **part of your code where a variable is accessible**. It determines where you can use the variable's name to refer to its value.
- There are two primary scopes in Python:
  - **Global Scope:** Variables defined outside any function are in the global scope. They are accessible throughout the entire program.
  - **Local Scope:** Variables defined inside a function are in the local scope. They are only accessible within that function.

Lifetime:

RTPNOTES.vercel.app

- Lifetime refers to the **duration for which a variable exists in memory**. It determines how long the memory allocated to the variable is reserved.
- The lifetime of a variable is closely tied to its scope. A variable's memory is released when it goes out of scope.

Example

```
def calculate_area(length, width):
  """Calculates the area of a rectangle."""
  area = length * width  # Local variable
  return area

# Global variable
message = "This message is accessible throughout the program."

# Main program
rectangle_length = 5
rectangle_width = 3

calculated_area = calculate_area(rectangle_length, rectangle_width)

print(message)  # Accessing global variable
print("Area of the rectangle:", calculated_area)

# This line would cause an error because 'length' and 'width' are local to
the function
# print("Length outside the function:", length)
# print("Width outside the function:", width)
```

- `message` is defined outside any function, so it has **global scope** and is accessible throughout the program. Its lifetime lasts until the program ends.
- `length`, `width`, and `area` are defined inside the `calculate_area` function, so they have **local scope**. They are only accessible within that function and their lifetime ends when the function execution finishes.

RTPNOTES.vercel.app

Python-Module-1-University-Questions-Part-B

1. Write a python program to generate the following type of pattern for the given N rows .

1

1 2

1 2 3

1 2 3 4

```
def generate_pattern(n):
  """
  Generates a pattern of increasing numbers for the given number of rows.

  Args:
      n: The number of rows in the pattern.
  """

  for i in range(1, n + 1):
    # Print numbers from 1 to the current row number
    for j in range(1, i + 1):
      print(j, end=" ")
    print()  # Move to the next line after each row

# Get the number of rows from the user
while True:
  try:
    n = int(input("Enter the number of rows: "))
    if n <= 0:
      raise ValueError("Please enter a positive integer for the number of
rows.")
    break  # Exit the loop if valid input is entered
  except ValueError as e:
    print(e)
    print("Please enter a valid positive integer.")

# Generate and print the pattern
```

RTPNOTES.vercel.app

```
generate_pattern(n)
```

2. *Write the python program to print all prime numbers less than 1000.*

```
for num in range(1,1001):
   if num > 1:
       for i in range(2,num):
           if (num % i) == 0:
               break
       else:
           print(f"{num} is a prime number!")
```

3. *Write a Python program to find distance between two points (x1,y1) and (x2,y2).*

- We will use this formula
- $$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
import math

def calculate_distance(x1, y1, x2, y2):
    distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return distance

# Example usage
x1, y1 = 1, 2
x2, y2 = 4, 6
distance = calculate_distance(x1, y1, x2, y2)
print(f"The distance between points ({x1}, {y1}) and ({x2}, {y2}) is
{distance}")
```

RTPNOTES.vercel.app

- Here we import math module and compute the square root using math.sqrt()

4. *Write a Python program to find the roots of a quadratic equation, ax2+ bx + c =0 .*

Consider the cases of both real and imaginary roots.

Quadratic Formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

discriminant $\Delta = b^2 - 4ac$:

If $\Delta > 0$, there are two distinct real roots.

If $\Delta = 0$, there is one real root (repeated root).

If $\Delta < 0$, there are two complex (imaginary) roots.

```python
import math

def find_roots(a, b, c):
    # Calculate the discriminant
    discriminant = b**2 - 4*a*c

    # Determine roots based on the discriminant
    if discriminant > 0:
        root1 = (-b + math.sqrt(discriminant)) / (2 * a)
        root2 = (-b - math.sqrt(discriminant)) / (2 * a)
        return (root1, root2)  # Both roots are real
    elif discriminant == 0:
        root = -b / (2 * a)
        return (root, root)  # Both roots are the same (real)
    else:
        return "The roots are imaginary."

# Example usage
a = 1
```

RTPNOTES.vercel.app

```python
b = -3
c = 2
roots = find_roots(a, b, c)
print(f"The roots of the quadratic equation are: {roots}")

a = 1
b = 2
c = 1
roots = find_roots(a, b, c)
print(f"The roots of the quadratic equation are: {roots}")

a = 1
b = 2
c = 5
roots = find_roots(a, b, c)
print(f"The roots of the quadratic equation are: {roots}")
```

5. *Write a Python program to check whether a number is Armstrong number or not.*

```python
def is_armstrong_number(n):
    # Convert the number to a string to easily get each digit
    digits = str(n)
    num_digits = len(digits)

    # Initialize the sum
    armstrong_sum = 0

    # Loop through each digit and add the digit raised to the power of
num_digits to the sum
    for digit in digits:
        armstrong_sum += int(digit) ** num_digits

    # Check if the sum is equal to the original number
    return armstrong_sum == n

# Example usage
number = 153
```

RTPNOTES.vercel.app

```python
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")

number = 9474
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")

number = 123
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

6. Write a Python program to display the sum of odd numbers between a programmer specified upper and lower limit.

```python
def sum_of_odd_numbers(lower_limit, upper_limit):
    # Initialize the sum to 0
    total_sum = 0

    # Loop through the range from lower_limit to upper_limit (inclusive)
    for number in range(lower_limit, upper_limit + 1):
        # Check if the number is odd
        if number % 2 != 0:
            total_sum += number

    return total_sum

# Example usage
lower_limit = int(input("Enter the lower limit: "))
upper_limit = int(input("Enter the upper limit: "))
```

RTPNOTES.vercel.app

```python
# Ensure the lower limit is less than or equal to the upper limit
if lower_limit > upper_limit:
    print("Invalid input: lower limit should be less than or equal to upper
limit.")
else:
    result = sum_of_odd_numbers(lower_limit, upper_limit)
    print(f"The sum of odd numbers between {lower_limit} and {upper_limit}
is: {result}")
```

*7. Given the value of x, write a Python program to evaluate the following series upto n terms:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

```python
import math

def evaluate_series(x, n):
    # Initialize the sum to 0
    series_sum = 0

    # Loop through each term from 0 to n-1
    for i in range(n):
        # Calculate the i-th term: x^i / i!
        term = (x ** i) / math.factorial(i)
        # Add the term to the series sum
        series_sum += term

    return series_sum

# Example usage
x = float(input("Enter the value of x: "))
n = int(input("Enter the number of terms n: "))

result = evaluate_series(x, n)
print(f"The value of the series up to {n} terms is: {result}")
```

RTPNOTES.vercel.app

8. Write a python program to find X^Y or pow(X,Y) without using standard function.

```python
def power(x, y):
    result = 1
    for _ in range(y):
        result = result * x
    return result

# Example usage
x = float(input("Enter the base (X): "))
y = int(input("Enter the exponent (Y): "))

result = power(x, y)
print(f"{x} raised to the power of {y} is: {result}")
```

9. Write a python program to generate the following type of pattern for the given N rows where N <= 26.

A
A B
A B C D
A B C D E

```python
def generate_pattern(rows):
    # Define a string of uppercase letters
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    # Loop over each row
    for i in range(rows):
        # Loop over each character in the row
        for j in range(i + 1):
            # Print the character corresponding to the column number
            print(letters[j], end=" ")
        print()  # Move to the next line after printing each row
```

RTPNOTES.vercel.app

```python
# Get user input for the number of rows
rows = int(input("Enter the number of rows (N <= 26): "))

# Call the function to generate the pattern
generate_pattern(rows)
```

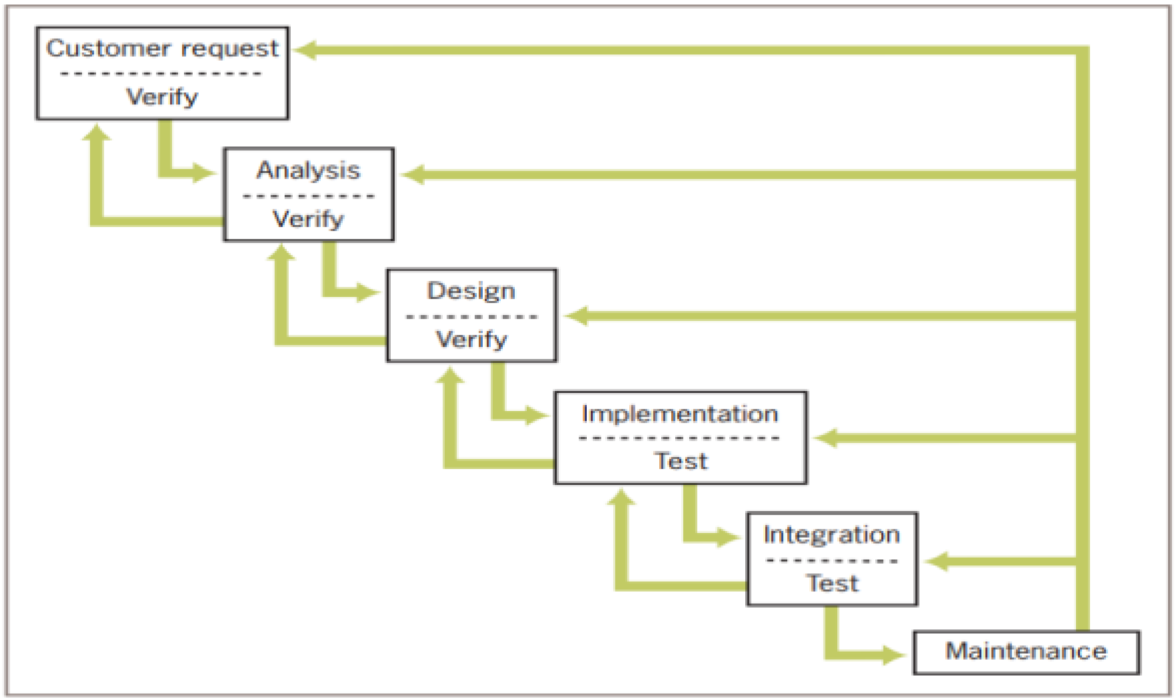
10. Discuss the steps involved in the waterfall model of software development process with the help of a neat diagram.

The waterfall model consists of several phases

1. Customer Request
    1. In this phase, the programmers receive a broad statement of a problem
2. Analysis
    1. The programmers determine what the program will do.
3. Design
    1. The programmers determine how the program will do its tasks
4. Implementation
    1. The programmers write the program
5. Integration
    1. Large programs have many parts. In the integration phase, these parts are broguht together into a smoothly functioning whole. Usually not an easy task
6. Maintenance
    1. Programs usually have a long life. A lifespan of 5 to 15 years is common for software, during this time, requirements change, errors are detected and minor/major modifications

RTPNOTES.vercel.app

are made.



11. Write a Python program to print all numbers between 100 and 1000 whose sum of digits is divisible by 9.

```python
def sum_of_digits(number):
    # Initialize sum to 0
    digit_sum = 0

    # Calculate the sum of digits
    while number > 0:
        digit_sum += number % 10
        number //= 10

    return digit_sum

# Iterate through numbers between 100 and 1000
for num in range(100, 1001):
    # Check if the sum of digits is divisible by 9
    if sum_of_digits(num) % 9 == 0:
        print(num)
```

RTPNOTES.vercel.app

12. Illustrate the use of range() in Python.

What is range?

The `range()` function in Python is used to generate a sequence of numbers. It's commonly used in `for` loops to iterate over a sequence of numbers. The `range()` function can take one, two, or three arguments:

1. If only one argument is provided, `range(stop)`, it generates a sequence of numbers from `0` to `stop - 1`.
2. If two arguments are provided, `range(start, stop)`, it generates a sequence of numbers from `start` to `stop - 1`.
3. If three arguments are provided, `range(start, stop, step)`, it generates a sequence of numbers from `start` to `stop - 1` with a step size of `step`.

Example

```python
for i in range(10):
    print(i, end=' ')
# Output: 0 1 2 3 4 5 6 7 8 9
```

```python
for i in range(5, 10):
    print(i, end=' ')
# Output: 5 6 7 8 9
```

```python
for i in range(2, 11, 2):
    print(i, end=' ')
# Output: 2 4 6 8 10
```

```python
numbers = list(range(5))
print(numbers)
# Output: [0, 1, 2, 3, 4]
```

```python
for i in range(10, 0, -1):
    print(i, end=' ')
```

RTPNOTES.vercel.app

```
# Output: 10 9 8 7 6 5 4 3 2 1
```

13. Write a Python program to print all prime factors of a given number.

```python
def prime_factors(number):
    factors = []
    divisor = 2

    while number > 1:
        while number % divisor == 0:
            factors.append(divisor)
            number //= divisor
        divisor += 1

    return factors

# Get user input for the number
num = int(input("Enter a number: "))

# Get the prime factors of the number
factors = prime_factors(num)

# Print the prime factors
if len(factors) == 0:
    print("No prime factors found.")
else:
    print(f"The prime factors of {num} are: {factors}")
```

14. Write a Python code to check whether a given year is a leap year or not

An year is a leap year if it's divisible by 4 but not divisible by 100 except for those divisible by 400.**

RTPNOTES.vercel.app

```python
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

# Get user input for the year
year = int(input("Enter a year: "))

# Check if the year is a leap year
if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

15. What are the possible errors in a Python program.

1. **SyntaxError**: Occurs when the Python parser encounters a syntax error in the code. This could be due to a missing parenthesis, colon, or quotation mark, or any other syntax rule violation.
2. **IndentationError**: Occurs when there is an issue with the indentation of code blocks. Python relies on indentation to define the structure of the code, so improper indentation can lead to errors.
3. **NameError**: Occurs when a variable or function name is used before it's defined or is misspelled.
4. **TypeError**: Occurs when an operation is performed on an object of inappropriate type. For example, trying to add a string and an integer, or accessing an index of a non-indexable object.
5. **IndexError**: Occurs when trying to access an index that is out of range for the given object, such as a list or a string.
6. **ValueError**: Occurs when a function receives an argument of the correct type but with an inappropriate value. For example, passing a negative value to the `math.sqrt()` function.
7. **KeyError**: Occurs when trying to access a key that does not exist in a dictionary.

RTPNOTES.vercel.app

8. **AttributeError**: Occurs when trying to access or use an attribute or method that does not exist for a given object.
9. **ImportError**: Occurs when Python cannot find the module to import or when there is an issue with the imported module itself.
10. **ZeroDivisionError**: Occurs when dividing a number by zero.
11. **FileNotFoundError**: Occurs when trying to open or access a file that does not exist.
12. **AssertionError**: Occurs when an `assert` statement fails.

16. Write a Python program to print the value of 2^(2n)+n+5 for n provided by the user.

```python
# Get user input for the value of n
n = int(input("Enter the value of n: "))

# Calculate the expression 2^(2n) + n + 5
result = 2**(2 * n) + n + 5

# Print the result
print(f"The value of 2^(2n) + n + 5 for n = {n} is: {result}")
```

17. Write a Python code to determine whether the given string is a Palindrome or not using slicing. Do not use any string function.

```python
def is_palindrome(s):
    return s == s[::-1]

# Get user input for the string
string = input("Enter a string: ")

# Check if the string is a palindrome
if is_palindrome(string):
    print("The string is a palindrome.")
```

RTPNOTES.vercel.app

```
else:
    print("The string is not a palindrome.")
```