Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 9

Roll No : 352

Name : Smit Gohel

01) Write a function to calculate BMI given mass and height. (BMI = mass/h**2)

In [3]:
```
def bmi(mass, height):
    return mass/height**2

mass = int(input('Enter mass: '))
height = float(input('Enter height: '))

print(f'BMI : {bmi(mass,height)}')
```

```
BMI : 0.012009757928316758
```

02) Write a function that add first n numbers.

In [6]:
```
def sum_of_first_n(n):
    return n * (n + 1) / 2

num = int(input('Enter n: '))
print(f'Sum of First n numbers: {sum_of_first_n(num)}')
```

```
Sum of First n numbers: 10.0
```

03) Write a function that returns 1 if the given number is Prime or 0 otherwise.

In [12]:
```
def is_Prime(n):
    if n == 1:
        return False
    else:
        for i in range(2, n):
```

```
            if n % i == 0:
                return False
    return True
num = int(input('Enter number: '))
print(f'{num} is Prime: {is_Prime(num)}')
```

```
7 is Prime: True
```

04) Write a function that returns the list of Prime numbers between given two numbers.

In [18]:
```
# def list_of_Prime_num(start, end):
#     res = []
#     for i in range(start+1, end):
#         if i != 1:
#             for j in range(2, i):
#                 if i % j == 0:
#                     return False
#                 else:
#                     res.append(i)

#     return res

def is_Prime(n):
    if n == 1:
        return False
    else:
        for i in range(2, n):
            if n % i == 0:
                return False
    return True

num1 = int(input('Enter starting number: '))
num2 = int(input('Enter ending number: '))
result = []

for i in range(num1+1, num2):
    if is_Prime(i):
        result.append(i)

print(f'List of Prime number between {num1}, {num2}: {result}')
```

```
List of Prime number between 0, 5: [2, 3]
```

05) Write a function that returns True if the given string is Palindrome or False otherwise.

In [19]:
```
def is_String_Palindrome(string):
    return string == string[::-1]

s = input('Enter String: ')
print(f'String is Palindrome : {is_String_Palindrome(s)}')
```

```
String is Palindrome : True
```

06) Write a function that returns the sum of all the elements of the list.

In [21]:
```
def sum_of_list_element(li):
    # sum = 0
    # for i in li:
    #     sum += i

    return sum(li)

li = list(map(int,input('Enter number: ').split()))
print(f'Sum of all the elements of list: {sum_of_list_element(li)}')
```

```
Sum of all the elements of list: 6
```

07) Write a function to calculate the sum of the first element of each tuples inside the list.

In [31]:
```
def sum_of_first_list_tuple_element(li):
    sum_0 = 0

    for i in li:
        sum_0 += i[0]

    return sum_0

n = int(input('Enter total number of list: '))
li = []
for _ in range(n):
    li.append(tuple(map(int,input('Enter elements: ').split())))

print(f'Sum of the first element of each tuples inside the list: {sum_of_first_l
```

```
Sum of the first element of each tuples inside the list: 3
```

08) Write a recursive function to find nth term of Fibonacci Series.

In [25]:
```
def nth_term_fibonacci(n):
    if n == 1:
        return 0
    if n == 2:
        return 1
    return nth_term_fibonacci(n-1) + nth_term_fibonacci(n-2)

n = int(input('Enter nth term: '))
print(f'nth term of Fibonacci Series: {nth_term_fibonacci(n)}')
```

```
nth term of Fibonacci Series: 3
```

09) Write a function to get the name of the student based on the given rollno.

Example: Given dict1 = {101:'Ajay', 102:'Rahul', 103:'Jay', 104:'Pooja'} find name of student whose rollno = 103

In [33]:
```
def find_student(dict1, rollno):
    for index, value in dict1.items():
        if rollno == index:
            return value
```

```
dict1 = {101:'Ajay', 102:'Rahul', 103:'Jay', 104:'Pooja'}
rollno = int(input('Enter roll no: '))
print(f'Name of student of {rollno}: {find_student(dict1, rollno)}')
```

```
Name of student of 103: Jay
```

10) Write a function to get the sum of the scores ending with zero.

Example : scores = [200, 456, 300, 100, 234, 678]

Ans = 200 + 300 + 100 = 600

In [1]:
```
def sum_of_ending_zero_elements(li):
    # return sum(i for i in li if i % 10 == 0)
    res = 0
    for i in li:
        if i % 10 == 0:
            res += i
    return res

scores = list(map(int,input('Enter number: ').split()))

print(f'Sum of the scores ending with zero: {sum_of_ending_zero_elements(scores)}')
```

```
Sum of the scores ending with zero: 600
```

11) Write a function to invert a given Dictionary.

hint: keys to values & values to keys

Before : {'a': 10, 'b':20, 'c':30, 'd':40}

After : {10:'a', 20:'b', 30:'c', 40:'d'}

In [2]:
```
def change_key_value(dict1):
    res = {}
    for key, value in dict1.items():
        res[value] = key

    return res

dict1 = {'a': 10, 'b':20, 'c':30, 'd':40}

print(f'{dict1}: {change_key_value(dict1)}')
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}: {10: 'a', 20: 'b', 30: 'c', 40: 'd'}
```

12) Write a function that returns the number of uppercase and lowercase letters in the given string.

example : Input : s1 = AbcDEfgh ,Ouptput : no_upper = 3, no_lower = 5

In [6]:
```
def count_letters(s):
    upper = 0; lower = 0
    for i in s:
```

```
        if i.isupper():
            upper += 1
        else:
            lower += 1

    return upper, lower

s = input('Enter string: ')
upper, lower = count_letters(s)
print(f'number of uppercase and lowercase letters in {s} : {upper} {lower}')
```

```
number of uppercase and lowercase letters in AbcDEfgh : 3 5
```

13) Write a lambda function to get smallest number from the given two numbers.

In [21]:
```
num1 = int(input('Enter num1: '))
num2 = int(input('Enter num2: '))
ans =(lambda num1, num2 : min(num1,num2)) (num1,num2)
print(ans)
```

```
5
```

14) For the given list of names of students, extract the names having more that 7 characters. Use filter().

In [1]:
```
li = list(map(str,input('Enter number: ').split()))

result = list(filter(lambda x: len(x) > 7, li))

print(list(result))
```

```
['smitttttt']
```

15) For the given list of names of students, convert the first letter of all the names into uppercase. use map().

In [6]:
```
li = list(map(str,input('Enter names of students: ').split()))

result = map(lambda name : name.title(), li)

print(list(result))
```

```
['Smit', 'Black']
```

16) Write udfs to call the functions with following types of arguments:

1. Positional Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable Legngth Positional(*args) & variable length Keyword Arguments (**kwargs)
5. Keyword-Only & Positional Only Arguments

In [4]:
```
def positional_args(name, age, city):
    """
```

```
    Function with positional arguments.
    Arguments must be passed in the correct order.
    """
    return f"{name} is {age} years old and lives in {city}."

# Example usage
result = positional_args("Alice", 25, "New York")
print(result)  # "Alice is 25 years old and lives in New York."

# This would cause an error
# result = positional_args(25, "Alice", "New York")  # Wrong order
# print(result)  # "Alice is 25 years old and lives in New York."
```

```
Alice is 25 years old and lives in New York.
```

In [1]:
```
def keyword_args(name, age, city):
    """
    Function with keyword arguments.
    Arguments are passed with parameter names.
    """
    return f"{name} is {age} years old and lives in {city}."

# Example usage
result = keyword_args(name="Alice", age=25, city="New York")
print(result)  # "Alice is 25 years old and lives in New York."

# Order doesn't matter with keyword arguments
result = keyword_args(city="New York", name="Alice", age=25)
print(result)  # Same result

# Mixed positional and keyword (positional must come first)
result = keyword_args("Alice", city="New York", age=25)
print(result)
```

```
Alice is 25 years old and lives in New York.
Alice is 25 years old and lives in New York.
Alice is 25 years old and lives in New York.
```

In [2]:
```
def default_args(name, age=30, city="Unknown"):
    """
    Function with default arguments.
    Default values are used when arguments are not provided.
    """
    return f"{name} is {age} years old and lives in {city}."

# Example usage
result1 = default_args("Alice")  # Uses defaults for age and city
print(result1)  # "Alice is 30 years old and lives in Unknown."

result2 = default_args("Bob", 25)  # Overrides age, uses default city
print(result2)  # "Bob is 25 years old and lives in Unknown."

result3 = default_args("Charlie", city="London")  # Uses default age
print(result3)  # "Charlie is 30 years old and lives in London."

# IMPORTANT: Mutable default argument pitfall
# def bad_default(arg=[]):
#     arg.append(1)
#     return arg

# def good_default(arg=None):
```

```
#     arg = arg or []
#     arg.append(1)
#     return arg
```

```
Alice is 30 years old and lives in Unknown.
Bob is 25 years old and lives in Unknown.
Charlie is 30 years old and lives in London.
```

In [3]:
```python
def variable_positional(*args):
    """
    Function with variable length positional arguments.
    All positional arguments are packed into a tuple.
    """
    print(f"Type of args: {type(args)}")  # tuple
    print(f"Number of arguments: {len(args)}")
    print(f"Arguments: {args}")

    # Process arguments
    total = 0
    for arg in args:
        if isinstance(arg, (int, float)):
            total += arg
    return total

# Example usage
sum1 = variable_positional(1, 2, 3)
print(f"Sum: {sum1}")  # Sum: 6

sum2 = variable_positional(10, 20, 30, 40, 50)
print(f"Sum: {sum2}")  # Sum: 150

sum3 = variable_positional()  # No arguments
print(f"Sum: {sum3}")  # Sum: 0
```

```
Type of args: <class 'tuple'>
Number of arguments: 3
Arguments: (1, 2, 3)
Sum: 6
Type of args: <class 'tuple'>
Number of arguments: 5
Arguments: (10, 20, 30, 40, 50)
Sum: 150
Type of args: <class 'tuple'>
Number of arguments: 0
Arguments: ()
Sum: 0
```

In [4]:
```python
def variable_keyword(**kwargs):
    """
    Function with variable length keyword arguments.
    All keyword arguments are packed into a dictionary.
    """
    print(f"Type of kwargs: {type(kwargs)}")  # dict
    print(f"Number of keyword arguments: {len(kwargs)}")
    print(f"Keyword arguments: {kwargs}")

    # Process arguments
    result = {}
    for key, value in kwargs.items():
        result[key.upper()] = value * 2 if isinstance(value, (int, float)) else
    return result
```

```
# Example usage
result1 = variable_keyword(name="Alice", age=25, city="NYC")
print(f"Processed: {result1}")

result2 = variable_keyword(x=10, y=20, z=30)
print(f"Processed: {result2}")

result3 = variable_keyword()  # No keyword arguments
print(f"Processed: {result3}")
```

```
Type of kwargs: <class 'dict'>
Number of keyword arguments: 3
Keyword arguments: {'name': 'Alice', 'age': 25, 'city': 'NYC'}
Processed: {'NAME': 'Alice', 'AGE': 50, 'CITY': 'NYC'}
Type of kwargs: <class 'dict'>
Number of keyword arguments: 3
Keyword arguments: {'x': 10, 'y': 20, 'z': 30}
Processed: {'X': 20, 'Y': 40, 'Z': 60}
Type of kwargs: <class 'dict'>
Number of keyword arguments: 0
Keyword arguments: {}
Processed: {}
```

In [5]:
```
def combined_args(a, b, c=10, *args, d=20, e=30, **kwargs):
    """
    Function combining all argument types.
    Order is important: positional, default, *args, keyword-only, **kwargs
    """
    print(f"Required positional: a={a}, b={b}")
    print(f"Default: c={c}")
    print(f"Variable positional (*args): {args}")
    print(f"Keyword-only defaults: d={d}, e={e}")
    print(f"Variable keyword (**kwargs): {kwargs}")

    total = a + b + c + d + e + sum(args) + sum(k for k in kwargs.values()
                                                if isinstance(k, (int, float)))
    return total

# Example usage
result = combined_args(1, 2, 3, 4, 5, 6, d=40, x=100, y=200, name="Alice")
print(f"Total: {result}")
```

```
Required positional: a=1, b=2
Default: c=3
Variable positional (*args): (4, 5, 6)
Keyword-only defaults: d=40, e=30
Variable keyword (**kwargs): {'x': 100, 'y': 200, 'name': 'Alice'}
Total: 391
```

In [6]:
```
def keyword_only_args(*, name, age, city="Unknown"):
    """
    Function with keyword-only arguments.
    All arguments after * must be passed as keyword arguments.
    """
    return f"{name} is {age} years old and lives in {city}."

# Example usage
result = keyword_only_args(name="Alice", age=25, city="NYC")
print(result)
```

```
# This would cause an error - all arguments must be keyword
# keyword_only_args("Alice", 25, "NYC")  # TypeError

# Partial keyword-only
def mixed_args(a, b, *, kw_only1, kw_only2):
    """First two can be positional, last two must be keyword."""
    return a + b + kw_only1 + kw_only2

result = mixed_args(1, 2, kw_only1=3, kw_only2=4)
print(result)  # 10
```

```
Alice is 25 years old and lives in NYC.
10
```

In [8]:

```
def positional_only_args(a, b, /, c, d, *, e, f):
    """
    Function with positional-only arguments.
    Arguments before / must be positional.
    Arguments after * must be keyword.
    Arguments between / and * can be either.
    """
    return f"a={a}, b={b}, c={c}, d={d}, e={e}, f={f}"

# Example usage
result = positional_only_args(1, 2, 3, d=4, e=5, f=6)
print(result)

# This would cause errors:
# positional_only_args(a=1, b=2, c=3, d=4, e=5, f=6)  # a,b must be positional
# positional_only_args(1, 2, 3, 4, 5, 6)  # e,f must be keyword
```

```
a=1, b=2, c=3, d=4, e=5, f=6
```