ASSIGNMENT 3 FUNCTION

```
Start coding or generate with AI.
```

1. What is the difference between a function and a method in Python?

-Function: A function is a block of code that performs a specific task and is defined independently of any class. It can be called directly by its name. Method: A method is also a block of code that performs a specific task, but it is associated with an object (an instance of a class). It is called on an object using the dot notation (e.g., object.method()).

2. Explain the concept of function arguments and parameters in Python.

- In Python, functions can accept inputs known as arguments. When defining a function, placeholders for these inputs are called parameters. When the function is called, the actual values passed in are the arguments.

  def add_numbers(a, b): # a and b are parameters sum = a + b return sum

  result = add_numbers(5, 3) # 5 and 3 are arguments print(result) # Output: 8

3. What are the different ways to define and call a function in Python?

- In Python, functions can be defined and called in several ways: Standard Function Definition This is the most common way to define a function, using the def keyword, followed by the function name, parentheses for arguments, and a colon. The function body is indented.

def greet(name): """This function greets the person passed in as a parameter.""" print("Hello, " + name + "!")

greet("World") # Calling the function

4. What is the purpose of the `return` statement in a Python function?

- The return statement in a Python function serves to terminate the function's execution and optionally provide a value back to the caller. When a return statement is encountered, the function immediately stops running, and the specified value, if any, is passed back as the result of the function call. If no value is explicitly specified, it implicitly returns None. The return statement enables functions to produce output that can be utilized in other parts of the program, facilitating modularity and code reusability.

def add(x, y): return x + y

result = add(5, 3) print(result) # Output: 8

5. What are iterators in Python and how do they differ from iterables?

- An iterator is an object that contains a countable number of values. An iterator is an object that can be iterated upon, meaning that you can traverse through all the values. Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods **iter**() and **next**() .

  Iterables are objects that can be looped over, while iterators are objects that provide a mechanism for traversing that loop. Every iterator is an iterable, but not every iterable is an iterator.

6. Explain the concept of generators in Python and how they are defined.

- A generator function is a special type of function that returns an iterator object. Instead of using return to send back a single value, generator functions use yield to produce a series of results over time. This allows the function to generate values and pause its execution after each yield, maintaining its state between iterations.

Basic Code Example: 1.def fun(max): 2.cnt = 1 3.while cnt <= max:

4. yield cnt
5. cnt += 1 6
6. ctr = fun(5)
7. for n in ctr:
8. print(n)
9. What are the advantages of using generators over regular functions?

- Generators offer several advantages over regular functions, primarily related to memory efficiency and lazy evaluation. Unlike regular functions that return a complete result set, generators produce values one at a time, allowing for efficient handling of large datasets or infinite sequences without exhausting memory. This "lazy evaluation" means processing only the data needed at the moment, and resuming from where the generator left off when called again.

1. Memory Efficiency
2. Lazy Evaluation
3. Easy Implementation
4. Representing Infinite Streams
5. Resource Pipelining
6. What is a lambda function in Python and when is it typically used?

- A lambda function, also known as an anonymous function, is a concise way to define a function within a single line. It's primarily used for short, simple operations where a full function definition using def would be overly verbose.

Higher-order functions: Lambda functions are frequently used as arguments to higher-order functions like map(), filter(), and sorted().

map(): Applies a function to all items in an iterable.

filter(): Filters an iterable based on a condition.

sorted(): Sorts a list based on a key (often a lambda function).

-

```
9.  Explain the purpose and usage of the `map()` function in Python.

-    The map() function in Python serves to apply a specified function
item within an iterable (such as a list, tuple, or set), and returns a
iterator that yields the results. It offers a concise way to perform
element-wise operations without explicit loops.

map(function, iterable, ...)

10.  What is the difference between `map()`, `reduce()`, and `filter()
functions in Python?

- map(), filter(), and reduce() are built-in higher-order functions in
that operate on iterables (like lists, tuples, etc.) but serve distinc
purposes:
map():
Applies a given function to each item in an iterable and returns an it
that yields the results. It transforms each element individually.
filter():
Creates a new iterator containing only the items from an iterable for
given function returns True. It selectively keeps elements based on a
reduce():
Applies a function cumulatively to the items of an iterable, from left
right, so as to reduce the iterable to a single value. It aggregates v
into one result.

from functools import reduce # Import reduce

numbers = [1, 2, 3, 4, 5]

# map(): squares each number
squared_numbers = map(lambda x: x**2, numbers)
print(list(squared_numbers)) # Output: [1, 4, 9, 16, 25]

# filter(): keeps only even numbers
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers)) # Output: [2, 4]

# reduce(): calculates the product of all numbers
product = reduce(lambda x, y: x * y, numbers)
print(product) # Output: 120

11. Using pen & Paper write the internal mechanism for sum operation u
reduce function on this given
list:[47,11,42,13];
-
from functools import reduce

# Function to add two numbers
def add(x, y):
    return x + y
```

9. Explain the purpose and usage of the `map()` function in Python.

- The map() function in Python serves to apply a specified function to every item within an iterable (such as a list, tuple, or set), and returns an iterator that yields the results. It offers a concise way to perform element-wise operations without explicit loops.

map(function, iterable, ...)

10. What is the difference between `map()`, `reduce()`, and `filter()` functions in Python?

- map(), filter(), and reduce() are built-in higher-order functions in Python that operate on iterables (like lists, tuples, etc.) but serve distinct purposes: map(): Applies a given function to each item in an iterable and returns an iterator that yields the results. It transforms each element individually. filter(): Creates a new iterator containing only the items from an iterable for which a given function returns True. It selectively keeps elements based on a condition. reduce(): Applies a function cumulatively to the items of an iterable, from left to right, so as to reduce the iterable to a single value. It aggregates values into one result.

from functools import reduce # Import reduce

numbers = [1, 2, 3, 4, 5]

map(): squares each number

squared_numbers = map(lambda x: x**2, numbers) print(list(squared_numbers)) # Output: [1, 4, 9, 16, 25]

filter(): keeps only even numbers

even_numbers = filter(lambda x: x % 2 == 0, numbers) print(list(even_numbers)) # Output: [2, 4]

reduce(): calculates the product of all numbers

product = reduce(lambda x, y: x * y, numbers) print(product) # Output: 120

```
a = [1, 2, 3, 4, 5]
res = reduce(add, a)

print(res)  # Output: 15
```

11. Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:[47,11,42,13];

- from functools import reduce

Function to add two numbers

def add(x, y): return x + y

a = [1, 2, 3, 4, 5] res = reduce(add, a)

print(res) # Output: 15

Practicle Question

1. Write a Python function that takes a list of numbers as input and returns the sum of all even numbers in the list.

Double-click (or enter) to edit

```
def sum_even_numbers(numbers):
    """
    Calculates the sum of all even numbers in a list.

    Args:
      numbers: A list of numbers.

    Returns:
      The sum of even numbers in the list.
    """
    even_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
    return even_sum
```

2. Create a Python function that accepts a string and returns the reverse of that string

```
def reverse_string(s):
    return s[::-1]
```

3. Implement a Python function that takes a list of integers and returns a new list containing the squares of each number.

```
def square_list(numbers):
    """
    Returns a new list containing the squares of each number in the input list.

    Args:
        numbers: A list of integers.

    Returns:
        A new list with the squares of the input numbers.
    """
    squared_numbers = []
    for number in numbers:
        squared_numbers.append(number**2)
    return squared_numbers
```

4. Write a Python function that checks if a given number is prime or not from 1 to 200

```
def is_prime(num):
    """
    Checks if a number is prime.

    A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

    Args:
        num: An integer to be checked for primality.

    Returns:
        True if num is prime, False otherwise.
    """
    if num <= 1:
        return False
    if num <= 3:
        return True
    if num % 2 == 0 or num % 3 == 0:
        return False
    i = 5
    while i * i <= num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True

def check_prime_range(limit):
  """
  Prints prime numbers from 1 to a specified limit.

  Args:
      limit: An integer representing the upper limit of the range.
  """
  for num in range(1, limit + 1):
      if is_prime(num):
          print(f"{num} is prime")
      else:
          print(f"{num} is not prime")

check_prime_range(200)
```

```
194 is not prime
```

```
194 is not prime
195 is not prime
196 is not prime
197 is prime
198 is not prime
199 is prime
200 is not prime
```

5. Create an iterator class in Python that generates the Fibonacci sequence up to a specified number of terms.

```
class FibonacciIterator:
    def __init__(self, limit):
        self.limit = limit
        self.current = 0
        self.next_num = 1
        self.count = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.count < self.limit:
            result = self.current
            self.current, self.next_num = self.next_num, self.current + self.next_num
            self.count += 1
            return result
        else:
            raise StopIteration

# Example usage:
fib_iter = FibonacciIterator(10)
for num in fib_iter:
    print(num)
```

```
0
1
1
2
3
5
8
13
21
34
```

6. Write a generator function in Python that yields the powers of 2 up to a given exponent.def powers_of_two(exponent): """ Generates powers of 2 up to a given exponent.

   Args:

   ```
   exponent: The maximum exponent.
   ```

   Yields:

   ```
   Successive powers of 2.
   ```

   """ for i in range(exponent + 1):

   ```
   yield 2 ** i
   ```

7. Implement a generator function that reads a file line by line and yields each line as a string.

```
def read_file_lines(filepath):
    """
    Reads a file line by line and yields each line.

    Args:
        filepath (str): The path to the file.

    Yields:
```

```
        str: Each line of the file.
    """
    try:
        with open(filepath, 'r') as file:
            for line in file:
                yield line.rstrip('\n')
    except FileNotFoundError:
        yield "File not found."

# Example usage:
filepath = 'my_file.txt'

# Create a dummy text file for demonstration
with open(filepath, 'w') as f:
    f.write("This is the first line.\n")
    f.write("This is the second line.\n")
    f.write("This is the third line.\n")

for line in read_file_lines(filepath):
    print(line)
```

```
This is the first line.
This is the second line.
This is the third line.
```

8. Use a lambda function in Python to sort a list of tuples based on the second --element of each tuple.

Double-click (or enter) to edit

```
list_of_tuples = [(1, 'z'), (2, 'a'), (3, 'b')]

sorted_list = sorted(list_of_tuples, key=lambda x: x[1])

print(sorted_list)
# Expected Output: [(2, 'a'), (3, 'b'), (1, 'z')]
```

```
[(2, 'a'), (3, 'b'), (1, 'z')]
```

9. Write a Python program that uses `map()` to convert a list of temperatures from Celsius to Fahrenheit.

```
c = 47

# Converting the temperature to Fahrenheit using the formula
f = (c * 1.8) + 32

print(str(c))
print(str(f))
```

```
47
116.60000000000001
```

10. Create a Python program that uses `filter()` to remove all the vowels from a given string.

```
# Python program to remove vowels from a string
# Function to remove vowels
def rem_vowel(string):
    vowels = ['a','e','i','o','u']
    result = [letter for letter in string if letter.lower() not in vowels]
    result = ''.join(result)
    print(result)

# Driver program
string = "out of subject - A Computer Science is the out of subject"
rem_vowel(string)
string = "Loving Python LOL"
rem_vowel(string)
```

```
t f sbjct -  Cmptr Scnc s th t f sbjct
Lvng Pythn LL
```

11. Imagine an accounting routine used in a book shop. It works on a list with sublists, which look like this:

Order number book title and author Quantity Price Per Item 34587 Learning Python Mark Lutz 4 40.95 98762 Programming python Mark Lutz 5 56.80 77226 Head First Python Paul Berry 3 32.95 88112 Einfuhrung in Python3, Bernd Klein 3 24.99

Write a Python program, which returns a list with 2-tuples. Each tuple consists of the order number and the product of the price per item and the quantity. The product should be increased by 10,- € if the value of the order is smaller than 100,00 €.

Write a Python program using lambda and map.

```
book_shop_data = [
    [34587, "Learning Python Mark Lutz", 4, 40.95],
    [98762, "Programming python Mark Lutz", 5, 56.80],
    [77226, "Head First Python Paul Berry", 3, 32.95],
    [88112, "Einfuhrung in Python3, Bernd Klein", 3, 24.99]
]

# Define a lambda function to calculate the total cost and adjust for small orders
calculate_total = lambda order: (
    order[0],  # Order number
    (order[2] * order[3]) + (10 if (order[2] * order[3]) < 100 else 0)
)

# Use map to apply the lambda function to each order
result = list(map(calculate_total, book_shop_data))

# Print the result
print(result)
```

```
[(34587, 163.8), (98762, 284.0), (77226, 108.85000000000001), (88112, 84.97)]
```

Start coding or generate with AI.