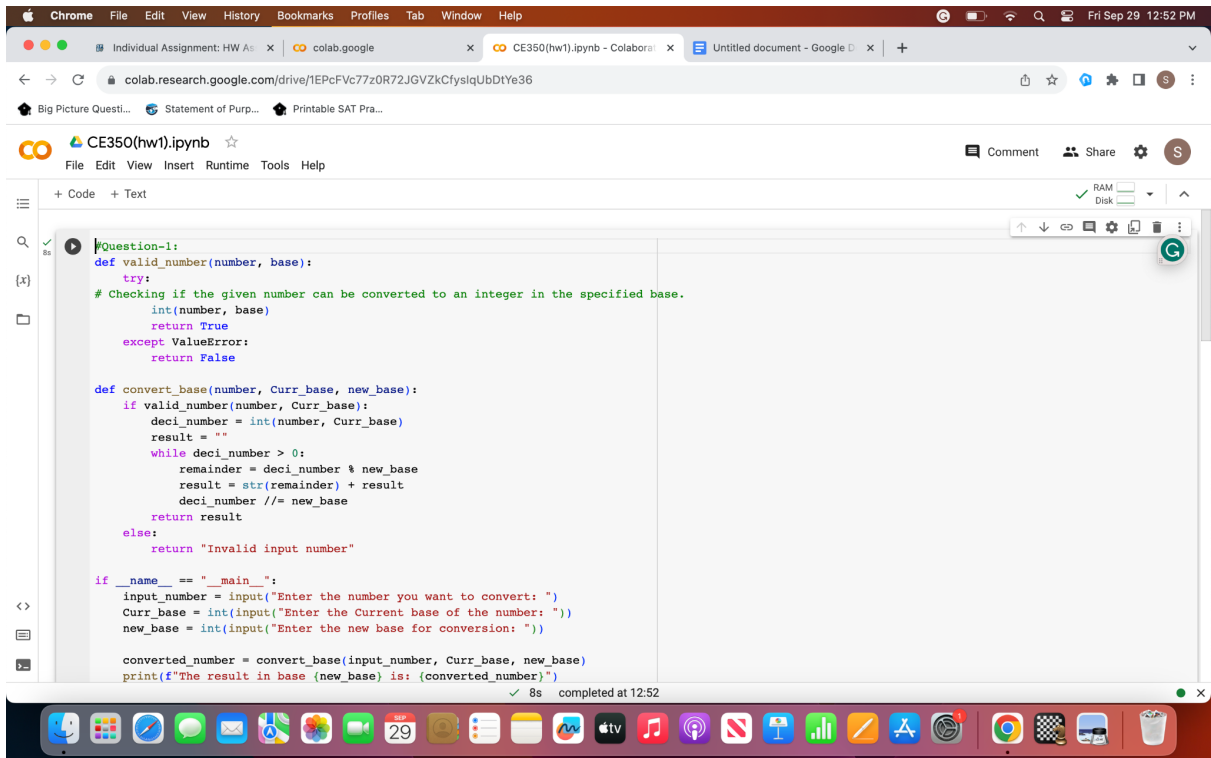


San Francisco Bay University.
Name- Swagat Neupane.
ID- 19698ns
Course- Computer Organization (CE305)



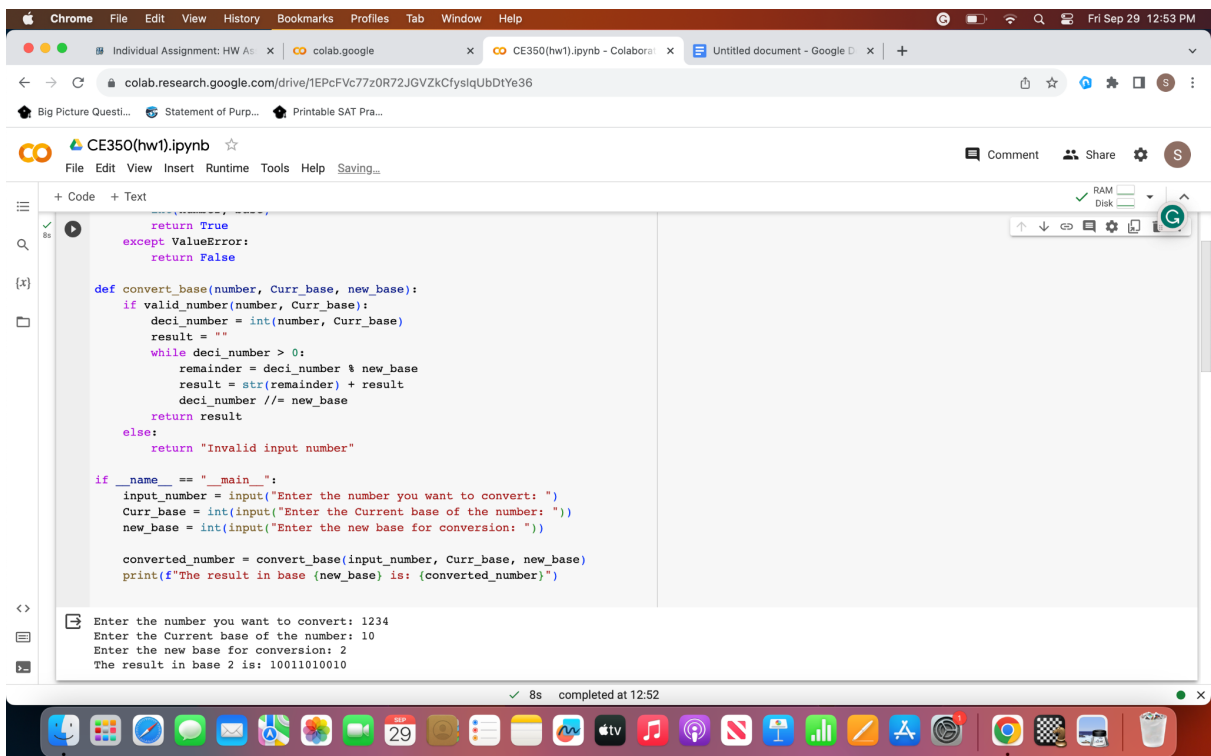
The screenshot shows a Jupyter Notebook titled "CE350(hw1).ipynb" in a Google Colaboratory environment. The code defines two functions: `valid_number` to check if a number is valid in a given base, and `convert_base` to convert a number from one base to another. The `convert_base` function uses a while loop to repeatedly divide the number by the new base and collect remainders. The main block prompts the user for input and prints the result.

```
#Question-1:
def valid_number(number, base):
    try:
        # Checking if the given number can be converted to an integer in the specified base.
        int(number, base)
        return True
    except ValueError:
        return False

def convert_base(number, Curr_base, new_base):
    if valid_number(number, Curr_base):
        deci_number = int(number, Curr_base)
        result = ""
        while deci_number > 0:
            remainder = deci_number % new_base
            result = str(remainder) + result
            deci_number //= new_base
        return result
    else:
        return "Invalid input number"

if __name__ == "__main__":
    input_number = input("Enter the number you want to convert: ")
    Curr_base = int(input("Enter the Current base of the number: "))
    new_base = int(input("Enter the new base for conversion: "))

    converted_number = convert_base(input_number, Curr_base, new_base)
    print(f"The result in base {new_base} is: {converted_number}")
```



This screenshot shows the same Jupyter Notebook after execution. The code is partially visible, and the output cell at the bottom shows the results of the program's execution based on the provided input.

```
return True
except ValueError:
    return False

def convert_base(number, Curr_base, new_base):
    if valid_number(number, Curr_base):
        deci_number = int(number, Curr_base)
        result = ""
        while deci_number > 0:
            remainder = deci_number % new_base
            result = str(remainder) + result
            deci_number //= new_base
        return result
    else:
        return "Invalid input number"

if __name__ == "__main__":
    input_number = input("Enter the number you want to convert: ")
    Curr_base = int(input("Enter the Current base of the number: "))
    new_base = int(input("Enter the new base for conversion: "))

    converted_number = convert_base(input_number, Curr_base, new_base)
    print(f"The result in base {new_base} is: {converted_number}")
```

Enter the number you want to convert: 1234
Enter the Current base of the number: 10
Enter the new base for conversion: 2
The result in base 2 is: 10011010010

Chrome File Edit View History Bookmarks Profiles Tab Window Help

CE350/CE350(hw1).ipynb at r... x Individual Assignment: HW As... x Copy of CE350(hw1).ipynb - C... x CE350/ at main - Swagaat/CE... x +

colab.research.google.com/drive/1NEUEy6MbnOCaRDmdEUZMsFaMG5_aSRIB

Big Picture Questi... Statement of Purp... Printable SAT Pra...

Copy of CE350(hw1).ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 17:53

+ Code + Text

```
def convert_to_floating_value(decimal_number):
    bits = 0
    bits = 0x2000 if decimal_number < 0 else 0x0000 # Set the sign bit (bit 13)

    exponent = 0
    normalized_decimal = abs(decimal_number)

    if normalized_decimal != 0:
        while normalized_decimal >= 2.0:
            normalized_decimal /= 2.0
            exponent += 1
        while normalized_decimal < 1.0:
            normalized_decimal *= 2.0
            exponent -= 1

    exponent += 6 # Bias for a 14-bit floating-point representation
    bits |= (exponent << 10) # Set the exponent bits (bits 12-10)

    # Extract the fractional part and convert to binary
    normalized_decimal -= 1.0
    fractional_part = normalized_decimal
    binary_fraction = []

    for i in range(10):
        fractional_part *= 2.0
        bit = int(fractional_part)
        binary_fraction.append(str(bit))
        fractional_part -= bit

    bits |= int(''.join(binary_fraction), 2) # Set the fractional part (bits 9-0)

    return bin(bits)[2:].zfill(14)

if __name__ == "__main__":
    given_decimal_number = 26.625 # User input
    binary_representation = convert_to_floating_value(given_decimal_number)
    print("The floating value for 14-bit binary:", binary_representation)
```

The floating value for 14-bit binary: 10101010101000

macOS dock with various application icons.

Chrome File Edit View History Bookmarks Profiles Tab Window Help

CE350/CE350(hw1).ipynb at r... x Individual Assignment: HW As... x Copy of CE350(hw1).ipynb - C... x CE350/ at main - Swagaat/CE... x +

colab.research.google.com/drive/1NEUEy6MbnOCaRDmdEUZMsFaMG5_aSRIB

Big Picture Questi... Statement of Purp... Printable SAT Pra...

Copy of CE350(hw1).ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 17:53

+ Code + Text

```
binary_fraction = []

for i in range(10):
    fractional_part *= 2.0
    bit = int(fractional_part)
    binary_fraction.append(str(bit))
    fractional_part -= bit

bits = int(''.join(binary_fraction), 2) # Set the fractional part (bits 9-0)

return bin(bits)[2:].zfill(14)

if __name__ == "__main__":
    given_decimal_number = 26.625 # User input
    binary_representation = convert_to_floating_value(given_decimal_number)
    print("The floating value for 14-bit binary:", binary_representation)
```

The floating value for 14-bit binary: 10101010101000

macOS dock with various application icons.