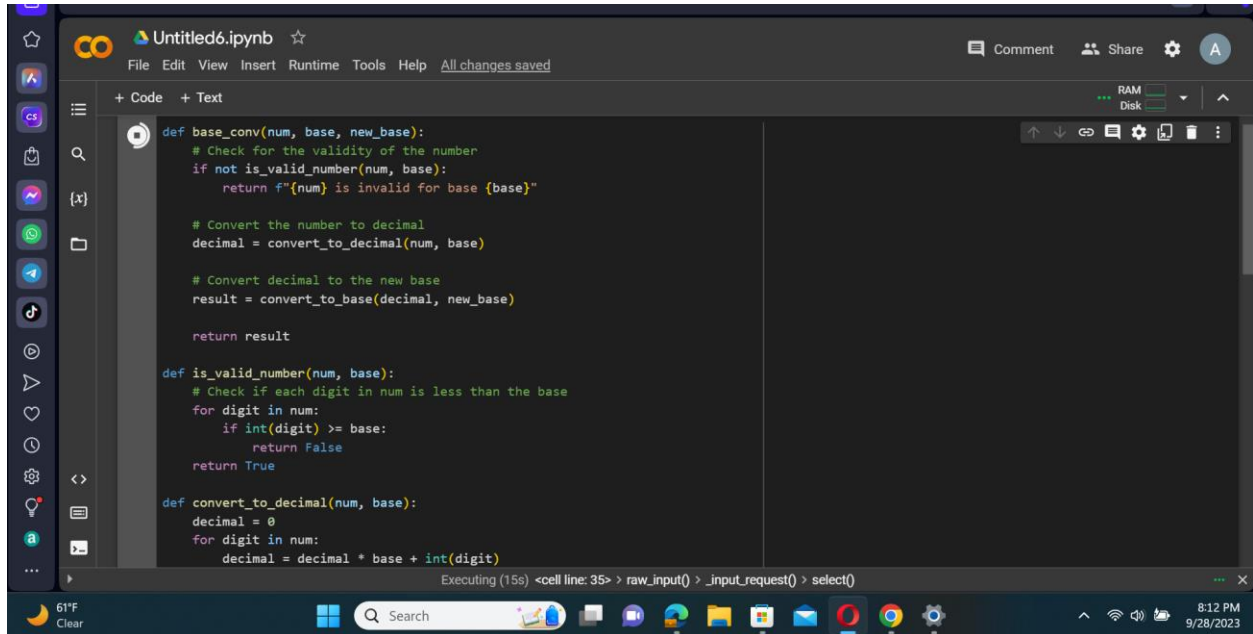
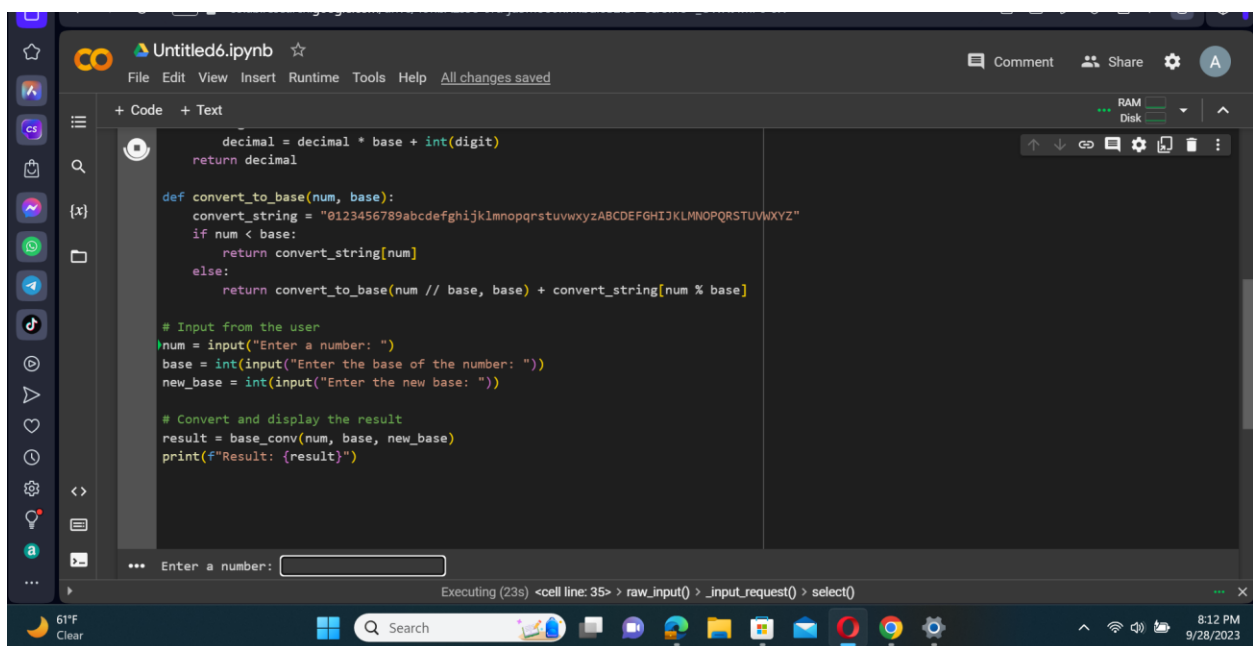


1. Write the program in any computer language to convert the given number from any base to a different base. The program needs to verify the validity of the given number first. If it is invalid, please prompt error information. Otherwise, print the correct result in the new base.



```
def base_conv(num, base, new_base):  
    # Check for the validity of the number  
    if not is_valid_number(num, base):  
        return f"{num} is invalid for base {base}"  
  
    # Convert the number to decimal  
    decimal = convert_to_decimal(num, base)  
  
    # Convert decimal to the new base  
    result = convert_to_base(decimal, new_base)  
  
    return result  
  
def is_valid_number(num, base):  
    # Check if each digit in num is less than the base  
    for digit in num:  
        if int(digit) >= base:  
            return False  
    return True  
  
def convert_to_decimal(num, base):  
    decimal = 0  
    for digit in num:  
        decimal = decimal * base + int(digit)
```

Executing (15s) <cell line: 35> > raw_input() > _input_request() > select()



```
        decimal = decimal * base + int(digit)  
    return decimal  
  
def convert_to_base(num, base):  
    convert_string = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"    
    if num < base:  
        return convert_string[num]  
    else:  
        return convert_to_base(num // base, base) + convert_string[num % base]  
  
# Input from the user  
num = input("Enter a number: ")  
base = int(input("Enter the base of the number: "))  
new_base = int(input("Enter the new base: "))  
  
# Convert and display the result  
result = base_conv(num, base, new_base)  
print(f"Result: {result}")
```

... Enter a number:

Executing (23s) <cell line: 35> > raw_input() > _input_request() > select()

2. Write the program in any computer language to convert the floating decimal number to 14-bits binary floating-point model as the real digital values in the hardware memory.

```
def floating_model(floating_dec):
    # Convert the decimal number to binary representation
    binary = bin(abs(int(floating_dec)))[2:] # Remove the "0b" prefix and convert to string
    binary = binary.zfill(5) # Pad the binary string with leading zeros to make it 5 bits long

    # Convert the fractional part to binary representation
    fractional_part = abs(floating_dec) - abs(int(floating_dec))
    fractional_binary = ""
    for _ in range(8):
        fractional_part *= 2
        bit = int(fractional_part)
        fractional_binary += str(bit)
        fractional_part -= bit

    # Determine the sign bit
    sign_bit = '1' if floating_dec < 0 else '0'

    # Combine the sign bit, integer part, and fractional part to form the 14-bit binary representation
    binary_representation = f"{sign_bit}_{binary}_{fractional_binary}"

    return binary_representation

# Ask the user for a floating-point decimal number
try:
```

Executing (10s) <cell line: 24> > raw_input() > _input_request() > select()

```
    # Determine the sign bit
    sign_bit = '1' if floating_dec < 0 else '0'

    # Combine the sign bit, integer part, and fractional part to form the 14-bit binary representation
    binary_representation = f"{sign_bit}_{binary}_{fractional_binary}"

    return binary_representation

# Ask the user for a floating-point decimal number
try:
    floating_dec = float(input("Enter a floating-point decimal number: "))
except ValueError:
    print("Invalid input. Please enter a valid floating-point decimal number.")
else:
    binary_representation = floating_model(floating_dec)
    print(f"14-bit binary representation: {binary_representation}")

*** Enter a floating-point decimal number: 
```

Executing (16s) <cell line: 24> > raw_input() > _input_request() > select()