

```

scores = [25,20,22,23,21,19]

def add_score(score):
    scores.append(score)

# Function to remove a specific score
def remove_score(score):
    if score in scores:
        scores.remove(score)
    else:
        print("Score not found!")

# Function to find the highest score
def get_highest_score():
    return max(scores) if scores else None

# Function to find the lowest score
def get_lowest_score():
    return min(scores) if scores else None

# Function to sort scores
def sort_scores():
    return sorted(scores)

# Function to calculate the average score
def get_average():
    return sum(scores) / len(scores) if scores else 0

```

```

print("All Scores:", scores)
print("Highest Score:", get_highest_score())
print("Lowest Score:", get_lowest_score())
print("Sorted Scores:", sort_scores())
print("Average Score:", get_average())

```

```

remove_score(20) # Removing a score
print("Scores after removal:", scores)

```

```

➤ All Scores: [25, 20, 22, 23, 21, 19]
  Highest Score: 25
  Lowest Score: 19
  Sorted Scores: [19, 20, 21, 22, 23, 25]
  Average Score: 21.666666666666668
  Scores after removal: [25, 22, 23, 21, 19]

```

```

from collections import namedtuple

```

```

# Define a namedtuple for Product
Product = namedtuple("Product", ["name", "price", "category"])

# Creating product instances
product1 = Product(name="Laptop", price=1200.99, category="Electronics")
product2 = Product(name="Sneakers", price=75.50, category="Footwear")

```

```

# Accessing product details
print("Product Name:", product1.name)
print("Price:", product1.price)
print("Category:", product1.category)

```

```

➤ Product Name: Laptop
  Price: 1200.99
  Category: Electronics

```

```

phonebook = {}

```

```

# Function to add a new contact
def add_contact(name, number):
    phonebook[name] = number
    print(f"Contact {name} added.")

# Function to update an existing contact
def update_contact(name, number):
    if name in phonebook:
        phonebook[name] = number
        print(f"Contact {name} updated.")
    else:
        print(f"Contact {name} not found!")

# Function to remove a contact
def remove_contact(name):
    if name in phonebook:
        del phonebook[name]
        print(f"Contact {name} removed.")
    else:
        print(f"Contact {name} not found!")

# Function to search for a contact
def search_contact(name):
    return phonebook.get(name, "Contact not found!")

# Function to check if a contact exists
def contact_exists(name):
    return name in phonebook

# Example Usage
add_contact("Alice", "123-456-7890")
add_contact("Bob", "987-654-3210")

print("Alice's Number:", search_contact("Alice"))
print("Bob's Number:", search_contact("anna"))

update_contact("Alice", "111-222-3333")
print("Updated Alice's Number:", search_contact("Alice"))

remove_contact("Bob")
print("Is Bob in phonebook?", contact_exists("Bob"))

➡ Contact Alice added.
Contact Bob added.
Alice's Number: 123-456-7890
Bob's Number: Contact not found!
Contact Alice updated.
Updated Alice's Number: 111-222-3333
Contact Bob removed.
Is Bob in phonebook? False

# Set to store unique emails
registered_emails = set()

# Function to add a new email
def add_email(email):
    if email in registered_emails:
        print(f"{email} is already registered.")
    else:
        registered_emails.add(email)
        print(f"{email} has been successfully registered.")

# Function to check if an email is already registered
def is_registered(email):
    return email in registered_emails

# Example Usage

```

```

add_email("user1@example.com")
add_email("user2@example.com")
add_email("user1@example.com") # Duplicate attempt

print("Is user1@example.com registered?", is_registered("user1@example.com"))
print("Is user3@example.com registered?", is_registered("user3@example.com"))

```

```

➞ user1@example.com has been successfully registered.
   user2@example.com has been successfully registered.
   user1@example.com is already registered.
   Is user1@example.com registered? True
   Is user3@example.com registered? False

```

```

def fibonacci_generator():
    a, b = 0, 1 # Starting values
    while True:
        yield a # Yield the next number
        a, b = b, a + b # Update values for the next step

# Using the generator to display the first 10 Fibonacci numbers
fib_gen = fibonacci_generator() # Create generator instance

for _ in range(10): # Get first 10 numbers
    print(next(fib_gen))

```

```

➞ 0
   1
   1
   2
   3
   5
   8
  13
  21
  34

```

```

# Function to calculate total earnings
def calculate_salary(hours_worked, hourly_rate, bonus=0):
    total_earnings = (hours_worked * hourly_rate) + bonus
    return total_earnings

# Example Usage
salary1 = calculate_salary(40, 20) # No bonus
salary2 = calculate_salary(45, 25, 100) # With a $100 bonus

print("Employee 1 Earnings:", salary1)
print("Employee 2 Earnings (with bonus):", salary2)

```

```

➞ Employee 1 Earnings: 800
   Employee 2 Earnings (with bonus): 1225

```

```

def calculate_ticket_price(age):
    standard_price = 10 # Fixed ticket price
    if age < 12:
        return standard_price * 0.5 # 50% discount for children
    elif age >= 60:
        return standard_price * 0.7 # 30% discount for seniors
    else:
        return standard_price # Full price for others

# Testing the function with different ages
test_ages = [5, 10, 25, 60, 75] # Child, young, adult, senior
for age in test_ages:
    print(f"Age {age}: Ticket Price = ${calculate_ticket_price(age):.2f}")

```

```

➞ Age 5: Ticket Price = $5.00
   Age 10: Ticket Price = $5.00

```

```

Age 25: Ticket Price = $10.00
Age 60: Ticket Price = $7.00
Age 75: Ticket Price = $7.00

```

```

# List of students with name, score, and age
students = [
    {"name": "Alice", "score": 85, "age": 18},
    {"name": "Bob", "score": 92, "age": 17},
    {"name": "Charlie", "score": 78, "age": 19},
    {"name": "David", "score": 88, "age": 18}
]

# Sorting by score (highest to lowest)
sorted_by_score = sorted(students, key=lambda x: x["score"], reverse=True)

# Sorting by age (youngest to oldest)
sorted_by_age = sorted(students, key=lambda x: x["age"])

# Display results
print("Sorted by Score (Highest to Lowest):")
for student in sorted_by_score:
    print(student)

print("\nSorted by Age (Youngest to Oldest):")
for student in sorted_by_age:
    print(student)

```

```

↔ Sorted by Score (Highest to Lowest):
{'name': 'Bob', 'score': 92, 'age': 17}
{'name': 'David', 'score': 88, 'age': 18}
{'name': 'Alice', 'score': 85, 'age': 18}
{'name': 'Charlie', 'score': 78, 'age': 19}

```

```

Sorted by Age (Youngest to Oldest):
{'name': 'Bob', 'score': 92, 'age': 17}
{'name': 'Alice', 'score': 85, 'age': 18}
{'name': 'David', 'score': 88, 'age': 18}
{'name': 'Charlie', 'score': 78, 'age': 19}

```

```

def factorial(n):
    if n == 1 or n == 0: # Base case
        return 1
    return n * factorial(n - 1) # Recursive step

```

```

# Testing with different numbers
numbers = [5, 7, 10]
for num in numbers:
    print(f"Factorial of {num}: {factorial(num)}")

```

```

↔ Factorial of 5: 120
   Factorial of 7: 5040
   Factorial of 10: 3628800

```

```

# List of product prices
prices = [50, 100, 75, 200, 150, 30]

# Function to apply a discount
def apply_discount(prices, discount_percent):
    return [price * (1 - discount_percent / 100) for price in prices]


# Function to find expensive items (above a certain threshold)
def find_expensive_items(prices, threshold):
    return [price for price in prices if price > threshold]

# Function to calculate total earnings
def total_earnings(prices):
    return sum(prices)

```

```
# Example usage
discounted_prices = apply_discount(prices, 10) # Apply 10% discount
expensive_items = find_expensive_items(discounted_prices, 80) # Find items above $80
total = total_earnings(discounted_prices) # Calculate total earnings

# Display results
print("Original Prices:", prices)
print("Discounted Prices:", discounted_prices)
print("Items Above $80:", expensive_items)
print("Total Earnings: $", total)
```

 Original Prices: [50, 100, 75, 200, 150, 30]
Discounted Prices: [45.0, 90.0, 67.5, 180.0, 135.0, 27.0]
Items Above \$80: [90.0, 180.0, 135.0]
Total Earnings: \$ 544.5