Yellow Set

Conceptual Questions

1. What is the difference between compiled and interpreted languages? Can you provide examples of each? What are the advantages/disadvantages of compiling vs interpreting? What kind of language is Python?
2. Explain the concept of version control and why it is important in software development.
3. Describe what an algorithm is and why algorithm efficiency is important. How can algorthm efficiency be described without referring to any specific implementations?

Programming Questions

Optionally, for each problem below, discuss the algorithm's complexity.

1. Create a function that takes a list of numbers and returns a new list with each number squared, maintaining the original order. Can you use a list comprehension?
2. Write a function to check if a given string contains only unique characters.
3. Given a list of integers, find the maximum product of two numbers in the list. What is the complexity of your solution? Can it be better?

blank

Yellow Set Answers

Conceptual (Yellow)

1. Compiled vs. Interpreted Languages:
   The primary difference between compiled and interpreted languages is in how code is translated to binary and executed.

- **Compiled Languages:**
  - The source code is translated into machine code by a compiler **before** the program runs. If the entire source code is compiled in advance of running, this is known as ahead-of-time (AOT) compilation. It's also possible to compile parts of the code after the application starts, which is referred to as just-in-time (JIT) compilation.
  - This produces a standalone executable file (or a collection of binary objects, e.g., packaged as dynamic libraries) that the hardware can run directly.
  - **Advantages:** Faster execution speed because the translation occurs ahead of time, optimized machine code, and often better performance.
  - **Disadvantages:** Longer development cycle for compile-debug-run iterations; less flexibility; platform-specific binaries that need recompilation for different systems.
  - **Examples:** C, C++, Rust, Go.
- **Interpreted Languages:**
  - The source code is translated line-by-line or statement-by-statement by an interpreter **at runtime**.
  - No separate executable is typically created; the program is run through the interpreter.
  - **Advantages:** Easier to test and debug; platform independence; flexible and dynamic.
  - **Disadvantages:** Slower execution speed because translation occurs during runtime; less efficient performance; potential security issues since source code is often visible.
  - **Examples:** Python, JavaScript, PHP.

What kind is Python?
Python is primarily an **interpreted language**. Its implementation involves a Python interpreter (such as CPython), which reads Python code and executes it directly. However, Python code is often compiled into bytecode (.pyc files) internally, which the interpreter then executes, but this compilation step is transparent to the user and not the same as to native machine code like C or C++.

2. Version Control:
   - Version control systems track changes to code, allowing teams to collaborate, maintain history, and revert changes.
   - Tools like Git enable features such as branching, merging, and managing project versions efficiently.
3. Algorithms and Efficiency:
   - An algorithm is a precise, step-by-step set of instructions designed to solve a specific problem or perform a particular task. It defines the logical sequence of operations necessary to achieve a desired outcome based on given inputs.
   - Why is algorithm efficiency important? Efficiency determines how quickly an algorithm can solve a problem, especially as the size of the input data increases. Efficient algorithms improve performance, reduce resource consumption (like CPU and memory), and make software scalable.

- How can algorthm efficiency be described without referring to any specific implementations? Algorithm efficiency can be described using abstract measures such as Big-O notation, which characterizes how the runtime or resource usage grows relative to the size of the input, regardless of specific implementation details. This provides a way to compare algorithms based on their theoretical scalability and performance in the worst or average case.

Programming (sample) answers (Yellow)

In each answer, make sure the interviewee also is able to provide a use case (test) and expected result. Watch for argument checks (i.e., not assuming valid args are passed, e.g., in #3).

Note that there are many possible correct answers.

```
# 1. Square each number in a list
def square_list(nums):
    return [n ** 2 for n in nums]

# Test
print(square_list([1, 2, 3, 4]))  # [1, 4, 9, 16]

# 2. Check for unique characters in a string
def is_unique(s):
    # It's fine to also solve with a loop or comprehension
    return len(set(s)) == len(s)

# Test
print(is_unique("abcde"))  # True
print(is_unique("hello"))  # False

# 3. Find maximum product of two numbers in a list
# Complexity: O(n^2), can be done in O(n) (discuss how)
def max_product(nums):
    if len(nums) < 2:
        raise ValueError("List must contain at least two numbers")
    max_prod = nums[0] * nums[1]
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            max_prod = max(max_prod, nums[i] * nums[j])
    return max_prod

# Test
print(max_product([1, 10, -5, 1, -100]))  # 10 * -5 = -50 (not max), 10 * 1 = 10, 1 * -100 = -100, etc.
print(max_product([2, 3, 4, 5]))  # 5 * 4 = 20
```

Green Set

Conceptual Questions

1. Explain the trade-offs between using iterative and recursive approaches in software design.
2. What does the DRY (Don't Repeat Yourself) principle mean, and why is it important in software development? Can you give a small code example?
3. What is the role of software testing, and what are some common types of tests employed in software development?

Programming Questions

Optionally, for each problem below, discuss the algorithm's complexity.

1. Given a list of integers, implement a function to find the index of the first occurrence of a specified element. Return -1 if not found.
2. Implement a function that takes a string and returns the same string with vowels removed.
3. Write a function to determine if a given number is a prime number.

blank

Green Set Answers

Conceptual (Green)

1. Iterative vs recursive approaches in software design.
    - Iterative approach: Uses loops (for, while) to repeat processes. Usually more memory-efficient and easier to understand for simple problems. Smaller, more elegant code, easier to maintain, bugs less likely.
    - Recursive approach: A function calls itself with smaller input to solve complex problems like tree traversals or divide-and-conquer algorithms. It can lead to elegant code but risks stack overflow if recursion depth is too deep, and generally uses more memory due to call stack overhead.
    - Choosing between them depends on problem complexity, performance needs, and readability.
2. DRY (Don't Repeat Yourself):
    - The DRY principle encourages developers to avoid duplicating code or logic within a codebase. Instead, code should be modular, reusable, and centralized to reduce errors and simplify maintenance.
    - Importance:
        - Minimizes the risk of bugs, because changes need to be made in only one place.
        - Makes the codebase cleaner and easier to understand.
        - Facilitates updates and future feature additions.
    - Example:

```
# Duplicated code
def process_user1(name):
    print(f"Processing user: {name}")

def process_user2(name):
    print(f"Processing user: {name}")
```

    You can refactor to a reusable function:

```
# Reusable function
def process_user(name):
    print(f"Processing user: {name}")

# Usage
process_user("Alice")
process_user("Bob")
```

3. Role of Software Testing:
    - Software testing ensures the software functions correctly and meets requirements.
    - Common types include unit testing, black-box testing, integration testing, system testing, and acceptance testing, each focusing on particular areas of the software lifecycle. It's fine to mention just one type and be able to explain what it is meant for.

Programming sample answers (Green)

In each answer, make sure the interviewee also is able to provide a use case (test) and expected result. Also watch for assumptions, e.g., in #1, whether the list is sorted or not is not specified.

Note that there are many possible correct answers.

```python
# 1. Find index of first occurrence
def find_first_index(lst, element):
    for i, val in enumerate(lst):
        if val == element:
            return i
    return -1

# Test
print(find_first_index([4, 5, 6, 7], 6))  # 2
print(find_first_index([1, 2, 3], 4))  # -1

# 2. Remove vowels from a string
def remove_vowels(s):
    vowels = "aeiouAEIOU"
    return ''.join(ch for ch in s if ch not in vowels)

# Test
print(remove_vowels("Hello World"))  # Hll Wrld
print(remove_vowels("Algorithm"))  # lgrthm

# 3. Check if a number is prime
import math

def is_prime(n):
    # Bonus: having extra special cases for efficiency
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False

    limit = int(math.sqrt(n))
    # A less efficient and more intuitive loop is fine
    for i in range(5, limit + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False
    return True

# Test
print(is_prime(37))  # True
print(is_prime(20))  # False
```