CPSC 1101 Introduction to Computing
Midterm Exam Guide

- **Exam Scope**: The entire material covered so far (week 1 – week 8 (Python List)).
- **Time limit**: 60 mins.
- **Question type**: multiple-choice, true/false, short answer.
- **Close book.**
- **MUST work independently! This is an individual-based exam.**
- **Exam date and time:** 10/25/2024 Friday, 11:00 AM - 12:00 PM
- **Format**: in-class and in-person exam

(The following are the suggested topics for review; however, it is not guaranteed that they will appear on the exam.)

This comprehensive guide will help you thoroughly prepare for your Python programming exam, covering key topics such as variables, data types, strings, operators, control statements, iterations, and lists. Each section includes detailed explanations, examples, common pitfalls, and practice exercises.

Table of Contents

Variables

Definition

- A **variable** in Python is a symbolic name that references or points to an object. It is a way to store data that can be used and manipulated throughout a program.

Variable Assignment

- Variables are created when you assign a value to them using the = operator.

```
python

x = 10
name = "Alice"
pi = 3.1416
```

- In Python, you do not need to declare a variable type; it is inferred from the value assigned.

Variable Naming Rules and Conventions

- **Rules** (must follow):

    - **Start with a letter or underscore**: Variable names must begin with a letter (a–z, A–Z) or an underscore (_).
    - **Case-sensitive**: Variable names are case-sensitive (`age` and `Age` are different).
    - **Allowed characters**: Letters, digits (0–9), and underscores.
    - **Cannot use reserved words**: Do not use Python keywords (e.g., `if`, `else`, `for`, `while`, `class`, `def`, etc.) as variable names.
- **Conventions** (should follow):
    - **Use descriptive names**: Variable names should be meaningful and describe the data they store.

      ```
      python

      total_price = 150.75
      is_valid = True
      ```

    - **Lowercase letters with underscores**: For variable names with multiple words, use underscores to separate them (snake_case).
    - **Constants in uppercase**: Variables intended to be constants should be in uppercase letters.

      ```
      python

      MAX_SIZE = 100
      ```

    - **Avoid single-letter names**: Except for common cases like `i`, `j`, `k` in loops.

Multiple Assignments

- Assign values to multiple variables in one line.

  ```
  python

  a, b, c = 5, 10, 15
  ```

- Assign the same value to multiple variables.

  ```
  python

  x = y = z = 0
  ```

Common Mistakes

- **Using undefined variables**: Ensure variables are defined before use.
- **Misnaming variables**: Watch out for typos in variable names.
- **Variable shadowing**: Be cautious when using the same variable name in different scopes.

Data Types

Python supports several built-in data types, which can be classified into the following categories:

Numeric Types

1. **Integers (`int`)**
   - Whole numbers, positive or negative, without decimals.

     ```
     python

     count = 100
     temperature = -15
     ```

2. **Floating Point Numbers (`float`)**
   - Numbers with decimal points.

     ```
     python

     price = 19.99
     pi = 3.1416
     ```

3. **Complex Numbers (`complex`)**
   - Numbers with a real and imaginary part, written as `a + bj`.

     ```
     python

     z = 2 + 3j
     ```

Sequence Types

1. **Strings (`str`)**
   - Ordered sequence of characters enclosed in single `' '`, double `" "`, or triple quotes `''' '''`/`""" """`.

     ```
     python

     greeting = "Hello, World!"
     ```

2. **Lists (`list`)**
   - Ordered, mutable collections of items.

     ```
     python

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

3. **Tuples (`tuple`)**
   - Ordered, immutable collections of items.

```
python

coordinates = (10.0, 20.0)
```

Mapping Types

- **Dictionaries (`dict`)**
  - Unordered collections of key-value pairs.

```
python

person = {"name": "Alice", "age": 30}
```

Set Types

1. **Sets (`set`)**
   - Unordered collections of unique items.

```
python

unique_numbers = {1, 2, 3, 4, 5}
```

2. **Frozen Sets (`frozenset`)**
   - Immutable version of sets.

Boolean Type

- **Boolean (`bool`)**
  - Represents `True` or `False`.

```
python

is_valid = True
has_errors = False
```

Binary Types

- **Bytes (`bytes`)**
- **Byte Arrays (`bytearray`)**
- **Memory View (`memoryview`)**

Special Type

- **NoneType**
  - Represents the absence of value (`None`).

```
python
```

```
result = None
```

Type Conversion (Type Casting)

- Converting data from one type to another using built-in functions:
  - `int()`, `float()`, `str()`, `bool()`, `list()`, `tuple()`, `set()`, etc.
- **Examples**:

```
python

# String to Integer
num_str = "10"
num_int = int(num_str)  # 10

# Integer to Float
num_float = float(num_int)  # 10.0

# List to Tuple
list_numbers = [1, 2, 3]
tuple_numbers = tuple(list_numbers)  # (1, 2, 3)
```

Checking Data Types

- Use the `type()` function to check the type of a variable.

```
python

print(type(num_int))  # <class 'int'>
```

Common Pitfalls

- **Integer Division**: In Python 3, dividing two integers with `/` results in a float.

```
python

result = 5 / 2  # 2.5
```

  - Use `//` for floor division (integer division).

```
python

result = 5 // 2  # 2
```

- **Type Errors**: Operations between incompatible types.

```
python

x = "10"
y = 5
z = x + y  # TypeError: can only concatenate str (not "int") to str
```

Strings

Definition

- A string is a sequence of characters used to store and represent text-based information.

String Creation

- Single quotes:

```
python

single_quote_str = 'Hello, World!'
```

- Double quotes:

```
python

double_quote_str = "Hello, World!"
```

- Triple quotes (for multi-line strings):

```
python

multi_line_str = '''This is a
multi-line string.'''
```

String Indexing and Slicing

- **Indexing**: Access individual characters using square brackets `[]`.
  - **Zero-based indexing**: The first character is at index `0`.

```
python

text = "Python"
first_char = text[0]  # 'P'
last_char = text[-1]  # 'n' (negative indexing starts from the
end)
```

- **Slicing**: Extract substrings using `[start:stop:step]`.

```
python

substring = text[1:4]  # 'yth' (characters at indices 1, 2, and 3)
```

- **Examples**:

```
python
```

```
text = "Hello, World!"
print(text[7:12])  # 'World'
print(text[:5])    # 'Hello' (from start to index 4)
print(text[7:])    # 'World!' (from index 7 to end)
print(text[::2])   # 'Hlo ol!' (every second character)
```

String Methods

- **Case Conversion**:
  - `upper()`: Convert to uppercase.

    ```
    python

    "hello".upper()  # 'HELLO'
    ```

  - `lower()`: Convert to lowercase.

    ```
    python

    "HELLO".lower()  # 'hello'
    ```

  - `title()`: Convert to title case.

    ```
    python

    "hello world".title()  # 'Hello World'
    ```

  - `capitalize()`: Capitalize the first character.

    ```
    python

    "python programming".capitalize()  # 'Python programming'
    ```

- **Whitespace Removal**:
  - `strip()`: Remove leading and trailing whitespace.

    ```
    python

    "  hello  ".strip()  # 'hello'
    ```

  - `lstrip()`: Remove leading whitespace.
  - `rstrip()`: Remove trailing whitespace.
- **Search and Replace**:
  - `find(substring)`: Return the index of the first occurrence.

    ```
    python

    "hello world".find("world")  # 6
    ```

  - `replace(old, new)`: Replace occurrences of a substring.

```
python

"hello world".replace("world", "Python")  # 'hello Python'
```

- **Splitting and Joining**:
  - `split(separator)`: Split a string into a list.

```
python

"one,two,three".split(",")  # ['one', 'two', 'three']
```

  - `join(iterable)`: Join elements of an iterable into a string.

```
python

", ".join(["apple", "banana", "cherry"])  # 'apple, banana,
cherry'
```

- **Validation Methods**:
  - `isdigit()`: Check if all characters are digits.
  - `isalpha()`: Check if all characters are alphabets.
  - `isalnum()`: Check if all characters are alphanumeric.
  - **Examples**:

```
python

"12345".isdigit()  # True
"abcde".isalpha()  # True
"abc123".isalnum()  # True
```

String Formatting

- **Old-style Formatting (`%`)**:

```
python

name = "Alice"
greeting = "Hello, %s!" % name  # 'Hello, Alice!'
```

- **`str.format()` Method**:

```
python

age = 30
message = "I am {} years old.".format(age)  # 'I am 30 years old.'
```

- **Formatted String Literals (f-strings)**: (Python 3.6+)

```
python

pi = 3.1416
```

```
print(f"The value of pi is approximately {pi:.2f}.")  # 'The value of
pi is approximately 3.14.'
```

Escape Characters

- Used to include special characters in strings.

Escape Sequence	Description
`\n`	New line
`\t`	Tab
`\'`	Single Quote
`\"`	Double Quote
`\\`	Backslash

- **Examples**:

```
python

print("Line1\nLine2")  # Output:
# Line1
# Line2

print("She said, \"Hello!\"")  # Output: She said, "Hello!"
```

String Operators

- **Concatenation (+)**:

```
python

full_name = "John" + " " + "Doe"  # 'John Doe'
```

- **Repetition (*)**:

```
python

separator = "-" * 10  # '----------'
```

- **Membership Operators**:
  - `in`: Check if a substring exists.

```
python

"apple" in "pineapple"  # True
```

  - `not in`: Check if a substring does not exist.

Common Mistakes

- **Immutable Nature of Strings**: Strings cannot be modified in place.

```
python

text = "Hello"
text[0] = "h"  # Error: 'str' object does not support item assignment
```

- **Concatenation of Different Types**: Cannot concatenate strings with non-strings without explicit conversion.

```
python

age = 25
message = "I am " + age + " years old."  # TypeError
# Correct way:
message = "I am " + str(age) + " years old."
```

Operators

Operators are special symbols that perform operations on variables and values.

Arithmetic Operators

Operator	Description	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

- **Examples**:

```
python

a = 15
b = 4

print(a + b)  # 19
print(a - b)  # 11
print(a * b)  # 60
print(a / b)  # 3.75
print(a % b)  # 3
print(a ** b) # 50625 (15 to the power of 4)
print(a // b) # 3
```

Assignment Operators

- Used to assign values to variables.

Operator	Example	Equivalent to
=	x = 5	
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
**=	x **= 3	x = x ** 3
//=	x //= 3	x = x // 3

- **Examples**:

```
python

x = 10
x += 5  # x = 15
x *= 2  # x = 30
```

Comparison Operators

- Compare two values and return a Boolean result (`True` or `False`).

Operator	Description	Example
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal	x >= y
<=	Less than or equal	x <= y

- **Examples**:

```
python

x = 10
y = 5

print(x == y)  # False
print(x != y)  # True
print(x > y)   # True
print(x < y)   # False
```

Logical Operators

- Combine conditional statements.

Operator	Description	Example
`and`	Logical AND	`x > 5 and x < 15`
`or`	Logical OR	`x < 5 or x > 15`
`not`	Logical NOT	`not(x == y)`

- **Examples**:

```
python

x = 10
y = 5

print(x > 5 and y < 10)  # True
print(x > 15 or y < 10)  # True
print(not(x == y))       # True
```

Identity Operators

- Compare objects to see if they are the same object with the same memory location.

Operator	Description	Example
`is`	Identical	`x is y`
`is not`	Not identical	`x is not y`

- **Examples**:

```
python

a = [1, 2, 3]
b = a
c = [1, 2, 3]

print(a is b)    # True
print(a is c)    # False
print(a == c)    # True
```

Membership Operators

- Test for membership in a sequence.

Operator	Description	Example
`in`	Exists in sequence	`x in y`

Operator	Description	Example
`not in`	Does not exist	`x not in y`

- **Examples**:

```
python

fruits = ["apple", "banana", "cherry"]

print("banana" in fruits)    # True
print("grape" not in fruits) # True
```

Operator Precedence

- Order in which operations are evaluated.
    1. Parentheses `()`
    2. Exponentiation `**`
    3. Unary plus and minus `+x`, `-x`
    4. Multiplication `*`, Division `/`, Floor Division `//`, Modulus `%`
    5. Addition `+`, Subtraction `-`
    6. Bitwise Shift `<<`, `>>`
    7. Bitwise AND `&`
    8. Bitwise XOR `^`
    9. Bitwise OR `|`
    10. Comparison Operators `==`, `!=`, `>`, `<`, `>=`, `<=`
    11. Assignment Operators `=`, `+=`, `-=`, etc.
- **Examples**:

```
python

result = 2 + 3 * 4  # 14 (3*4=12; 12+2=14)
result = (2 + 3) * 4  # 20 (2+3=5; 5*4=20)
```

Common Mistakes

- **Integer Division in Python 2 vs Python 3**: In Python 2, dividing two integers performs floor division, but in Python 3, it performs true division.

```
python

# Python 2
print(5 / 2)  # 2

# Python 3
print(5 / 2)  # 2.5
```

- **Operator Overloading**: Be aware that some operators behave differently with different types.

```
python

# Addition
print(1 + 2)          # 3
print("a" + "b")      # 'ab'

# Multiplication
print(3 * 4)          # 12
print("a" * 3)        # 'aaa'
```

Control Statements

Control statements determine the flow of execution based on conditions.

Conditional Statements

if Statement

- **Syntax**:

```
python

if condition:
    # code to execute if condition is True
```

- **Example**:

```
python

age = 20
if age > 18:
    print("You are older than 18.")
```

if-else Statement

- **Syntax**:

```
python

if condition:
    # code if True
else:
    # code if False
```

- **Example**:

```
python

if temperature >= 70:
    print("You may swim.")
```

```
else:
    print("You may not swim.")
```

if-elif-else Statement

- **Syntax**:

```
python

if condition1:
    # code if condition1 is True
elif condition2:
    # code if condition2 is True
else:
    # code if both conditions are False
```

- **Example**:

```
python

score = 85
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
else:
    grade = 'D'
```

Nested if Statements

- You can nest `if` statements within each other.

```
python

if condition1:
    if condition2:
        # code if both conditions are True
```

- **Example**:

```
python

if age >= 18:
    if has_license:
        print("You can drive.")
    else:
        print("You need a license to drive.")
else:
    print("You are too young to drive.")
```

Ternary Operator

- Shorthand for `if-else` statement.
- **Syntax**:

```
python

variable = true_value if condition else false_value
```

- **Example**:

```
python

max_value = x if x > y else y
```

Switch Statements

- Python does not have a built-in `switch` statement, but you can use dictionaries or `if-elif-else` chains.

Iterations

Iterations allow you to execute a block of code multiple times.

for Loop

- Used to iterate over a sequence (e.g., list, tuple, string).
- **Syntax**:

```
python

for variable in sequence:
    # code to execute
```

- **Examples**:

  Iterating over a List:

```
python

fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

  Using `range()`:

```
python

for i in range(5):
    print(i)  # Outputs 0 to 4
```

Range Variations:

```
python

range(stop)
range(start, stop)
range(start, stop, step)
```

- **Examples**:

```
python

for i in range(1, 6):
    print(i)  # Outputs 1 to 5

for i in range(0, 10, 2):
    print(i)  # Outputs 0, 2, 4, 6, 8
```

while Loop

- Repeats as long as a condition is true.
- **Syntax**:

```
python

while condition:
    # code to execute
```

- **Example**:

```
python

count = 0
while count < 5:
    print(count)
    count += 1
```

Loop Control Statements

1. **break**
   - Exits the loop immediately.
   - **Example**:

```
python

for i in range(10):
```

```
        if i == 5:
            break
        print(i)  # Outputs 0 to 4
```

2. **continue**
    - Skips the current iteration and continues with the next one.
    - **Example**:

```
python

for i in range(10):
    if i % 2 == 0:
        continue
    print(i)  # Outputs odd numbers between 0 and 9
```

3. **pass**
    - Does nothing; acts as a placeholder.
    - **Example**:

```
python

for i in range(5):
    pass  # Placeholder for future code
```

Nested Loops

- Loops inside other loops.
- **Example**:

```
python

for i in range(3):
    for j in range(2):
        print(f"i={i}, j={j}")
```

Iterating Over Dictionaries

- **Example**:

```
python

person = {"name": "Alice", "age": 30}
for key in person:
    print(f"{key}: {person[key]}")
```

Else Clause in Loops

- The `else` clause executes after the loop finishes normally (not terminated by `break`).
- **Example**:

```
python

for i in range(5):
    print(i)
else:
    print("Loop completed.")
```

Common Mistakes

- **Infinite Loops**: Forgetting to update the loop variable in a `while` loop.

```
python

count = 0
while count < 5:
    print(count)
    # Missing count += 1
```

- **Off-by-One Errors**: Misunderstanding the range of indices in loops.

Lists

Definition

- A list is an ordered, mutable collection of items.
- **Creation**:

```
python

empty_list = []
numbers = [1, 2, 3, 4, 5]
mixed_list = [1, "two", 3.0, True]
```

Accessing List Items

- **Indexing**:

```
python

first_item = numbers[0]
last_item = numbers[-1]
```

- **Slicing**:

```
python

sublist = numbers[1:4]  # Items at indices 1, 2, 3
```

Modifying Lists

- **Adding Items**:
  - `append(item)`: Add to the end.

    ```
    python

    numbers.append(6)
    ```

  - `insert(index, item)`: Insert at a specific position.

    ```
    python

    numbers.insert(2, 2.5)
    ```

  - `extend(iterable)`: Extend the list with elements from an iterable.

    ```
    python

    numbers.extend([7, 8, 9])
    ```

- **Removing Items**:
  - `remove(item)`: Remove the first occurrence of the item.

    ```
    python

    numbers.remove(2)
    ```

  - `pop(index)`: Remove and return item at index (default last item).

    ```
    python

    last_item = numbers.pop()
    ```

  - `clear()`: Remove all items.

    ```
    python

    numbers.clear()
    ```

- **Updating Items**:

  ```
  python

  numbers[0] = 0
  ```

List Methods

- **Counting and Searching**:
  - `count(item)`: Return the number of occurrences.

```
python

numbers.count(2)
```

- `index(item)`: Return the index of the first occurrence.

```
python

numbers.index(3)
```

- **Sorting and Reversing**:
  - `sort()`: Sort the list in ascending order.

```
python

numbers.sort()
```

  - `reverse()`: Reverse the elements of the list.

```
python

numbers.reverse()
```

List Comprehensions

- Compact way to create lists.

```
python

squares = [x ** 2 for x in range(10)]
even_numbers = [x for x in range(20) if x % 2 == 0]
```

Nested Lists

- Lists within lists (multi-dimensional lists).

```
python

matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
```

- **Accessing Nested List Items**:

```
python

element = matrix[1][2]  # 6
```

List Copying

- **Copy**:
  - Using slicing:

    ```
    python

    new_list = old_list[:]
    ```

  - Using `list()` constructor:

    ```
    python

    new_list = list(old_list)
    ```

  - Using `copy()` method:

    ```
    python

    new_list = old_list.copy()
    ```

Common Pitfalls

- **Modifying a List While Iterating Over It**: This can lead to unexpected behavior.

```
python

for item in my_list:
    if condition:
        my_list.remove(item)  # May skip items
```

  - **Solution**: Iterate over a copy or use list comprehension.

    ```
    python

    my_list = [item for item in my_list if not condition]
    ```

- **Index Out of Range**: Accessing indices beyond the list length.

Practice Exercises

Variables and Data Types

1. **Exercise**: Assign the following values to variables with appropriate names and types:
   - The integer number of days in a week.
   - The float value of pi up to 5 decimal places.
   - A boolean indicating if the sky is blue.
   - A string containing your full name.

Solution:

```
python

days_in_week = 7
pi_value = 3.14159
is_sky_blue = True
full_name = "John Doe"
```

2. **Exercise**: Convert the string `"100"` to an integer and add it to the integer `50`.

   Solution:

```
python

num_str = "100"
total = int(num_str) + 50  # 150
```

Strings

1. **Exercise**: Given the string `"Data Science"`, extract and print the substring `"Science"`.

   Solution:

```
python

text = "Data Science"
substring = text[5:]
print(substring)  # 'Science'
```

2. **Exercise**: Write a program that takes a user's first and last name and prints them in reverse order with a space between them.

   Solution:

```
python

first_name = input("Enter your first name: ")
last_name = input("Enter your last name: ")
print(f"{last_name} {first_name}")
```

Operators

1. **Exercise**: Write a program that calculates the area of a circle given the radius input by the user.

   Solution:

```
python
```

```
import math
radius = float(input("Enter the radius of the circle: "))
area = math.pi * radius ** 2
print(f"The area of the circle is {area}")
```

2. **Exercise**: Determine if a number is even or odd using the modulus operator.

   Solution:

   ```
   python

   number = int(input("Enter a number: "))
   if number % 2 == 0:
       print("Even")
   else:
       print("Odd")
   ```

Control Statements

1. **Exercise**: Write a program that checks if a given year is a leap year.

   Solution:

   ```
   python

   year = int(input("Enter a year: "))
   if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
       print(f"{year} is a leap year.")
   else:
       print(f"{year} is not a leap year.")
   ```

2. **Exercise**: Create a grading system that assigns a letter grade based on a numeric score.

   Solution:

   ```
   python

   score = float(input("Enter your score: "))
   if score >= 90:
       grade = 'A'
   elif score >= 80:
       grade = 'B'
   elif score >= 70:
       grade = 'C'
   elif score >= 60:
       grade = 'D'
   else:
       grade = 'F'
   print(f"Your grade is {grade}.")
   ```

Iterations

1. **Exercise**: Print the Fibonacci sequence up to `n` terms.

   Solution:

   ```
   python

   n_terms = int(input("How many terms? "))
   n1, n2 = 0, 1
   count = 0
   while count < n_terms:
       print(n1, end=' ')
       nth = n1 + n2
       n1 = n2
       n2 = nth
       count += 1
   ```

2. **Exercise**: Use a `for` loop to print the multiplication table of a given number.

   Solution:

   ```
   python

   num = int(input("Enter a number: "))
   for i in range(1, 11):
       print(f"{num} x {i} = {num * i}")
   ```

Lists

1. **Exercise**: Given a list of numbers, create a new list containing only the even numbers.

   Solution:

   ```
   python

   numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
   even_numbers = [num for num in numbers if num % 2 == 0]
   print(even_numbers)  # [2, 4, 6, 8, 10]
   ```

2. **Exercise**: Write a program to find the largest and smallest number in a list.

   Solution:

   ```
   python

   numbers = [23, 1, 45, 34, 7, 89, 2]
   largest = max(numbers)
   smallest = min(numbers)
   print(f"Largest: {largest}, Smallest: {smallest}")
   ```

Additional Tips for Exam Preparation

- **Practice Coding by Hand**: Write code on paper to prepare for written exams.
- **Understand Error Messages**: Familiarize yourself with common errors and exceptions.
- **Review Built-in Functions**: Know commonly used functions like `len()`, `range()`, `type()`, etc.
- **Time Management**: Allocate time wisely during the exam; start with questions you find easier.
- **Read Questions Carefully**: Ensure you understand what is being asked before writing code.
- **Test Your Code**: If possible, test your code with sample inputs to check for correctness.