CS101 Fall 2025 :: Midterm Preparation Guide

Python Programming Fundamentals

Table of contents

1 Overview of Midterm

This study guide covers all the essential Python concepts you need to know for the midterm exam in our course. During lab time on Thursday 23th October 2025. The exam will be **70 minutes long**, **closed-book**, and will test your understanding of Python fundamentals and concepts taken from the `Materials` pages in our course (see URL; https://cmpsc101fall2025datastructures.github.io/site/materials/0_materials.html. In addition, this midterm will cover chapters 1,2,3,4 and 5 of our textbook; *Introduction to Computation and Programming Using Python* by John V. Guttag.

1.1 Exam Format

What will the midterm look like? The below list provides a a general idea of what to expect.

- **Code Output Questions**: You will be given Python code and asked what it prints
- **Multiple Choice**: Select the best answer from given options

- **True/False with Explanation**: State whether something is true or false and explain why
- **Fill-in-the-Blank**: Complete code to achieve a specific result
- **Short Answer**: Explain concepts or differences between approaches
- **Guttag's textbook**: Chapters 1,2,3,4 and 5

1.2 Study Tips

How can you prepare for this midterm? Your instructor has assembled a non-exhaustive short list of some of the best ways to prepare for the midterm.

1. **Practice writing code by hand** - You will not have a computer during the exam
2. **Trace through code step by step** - Follow the execution mentally
3. **Know the exact output format** - Pay attention to spacing, quotes, brackets
4. **Understand concepts, don't just memorize** - Know why things work the way they do
5. **Review common errors** - Understanding what goes wrong helps you get it right

---

2 Key Concepts by Topic

Since the midterm questions draw upon lots of diverse material from the course, it might be helpful to your studying to follow the below format. Studying concepts in these group will help you to organize your thoughts for each type of question presented in the midterm. It is strongly recommended that you go carefully work through each of the sections by reading the examples, working the code and understanding how the code demonstrations the concepts at play.

2.1 1. Python Literals and Data Types

2.1.1 What You Need to Know

- **Literals** are fixed values written directly in code
- **Data types**: `int`, `float`, `str`, `bool`
- **Type conversion**: `int()`, `float()`, `str()`, `bool()`

2.1.2 Key Points

```
# Valid literals
42          # integer
3.14        # float
"hello"     # string
'world'     # string (single quotes also work)
True        # boolean
False       # boolean

# Invalid literals
3.14.5      # Two decimal points
True.       # Period after boolean
```

2.1.3 Practice Questions

Practice 1: Which of these are valid Python literals?

```
a) 100    b) 2.5.7    c) "Python"    d) True    e) 'coding'
```

Practice 2: What data type is each of these?

```
x = 25
y = 3.14
z = "Hello"
w = True
```

2.2 2. Variables and Operators

2.2.1 What You Need to Know

- **Assignment operator**: `=` (assigns value to variable)
- **Arithmetic operators**: `+`, `-`, `*`, `/`, `**` (power), `%` (modulus)
- **Comparison operators**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical operators**: `and`, `or`, `not`

2.2.2 Key Points

```
# Assignment vs Equality
x = 5       # Assignment (single =)
x == 5      # Comparison (double ==)

# Order of operations (PEMDAS)
result = 2 + 3 * 4      # Result is 14, not 20
result = (2 + 3) * 4    # Result is 20

# Modulus operator
7 % 3       # Result is 1 (remainder)
8 % 2       # Result is 0 (no remainder)
```

2.2.3 Practice Questions

Practice 3: What will this code output?

```
a = 10
b = 3
print(a + b)
print(a ** b)
print(a % b)
```

Practice 4: What will this expression evaluate to?

```
x = 8
result = x > 5 and x < 15
```

2.3 3. Conditionals (if/elif/else)

2.3.1 What You Need to Know

- **if statements** execute code when condition is `True`
- **elif** (else if) provides additional conditions to check
- **else** runs when no conditions are `True`
- **Indentation matters** - Use consistent spacing (usually 4 spaces)

2.3.2 Key Points

```
# Basic structure
if condition1:
    # Code runs if condition1 is True
elif condition2:
    # Code runs if condition2 is True (and condition1 was False)
else:
    # Code runs if all conditions were False

# Logical operators in conditions
if x > 0 and x < 10:        # Both must be true
if x < 0 or x > 100:        # Either can be true
if not x == 0:              # Same as x != 0
```

2.3.3 Practice Questions

Practice 5: What will this code print if `score = 85`?

```
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
else:
    print("F")
```

Practice 6: Complete this code to check if a number is even or odd:

```
number = 7
if __________:
    print("Even")
else:
    print("Odd")
```

2.4 4. Loops (for and while)

2.4.1 What You Need to Know

- **for loops** iterate over sequences (ranges, lists, strings)
- **while loops** continue as long as condition is `True`
- **range()** function: `range(stop)`, `range(start, stop)`, `range(start, stop, step)`

2.4.2 Key Points

```
# For loops with range
for i in range(5):          # 0, 1, 2, 3, 4
for i in range(2, 8):       # 2, 3, 4, 5, 6, 7
for i in range(1, 10, 2):   # 1, 3, 5, 7, 9

# While loops
```

```python
count = 0
while count < 5:
    print(count)
    count += 1               # Same as count = count + 1
```

2.4.3 Practice Questions

Practice 7: What will this for loop print?

```python
for i in range(2, 7, 2):
    print(i)
```

Practice 8: Complete this while loop to print numbers 5 down to 1:

```python
num = 5
while __________:
    print(num)
    __________
```

2.5 5. Strings and String Operations

2.5.1 What You Need to Know

- **String indexing**: `string[0]` is first character, `string[-1]` is last
- **String slicing**: `string[start:end:step]`
- **String methods**: `.lower()`, `.upper()`, `.replace()`, `.split()`, `.strip()`
- **F-strings**: `f"Hello {name}"`

2.5.2 Key Points

```python
text = "Python"
# Indexing (starts at 0)
text[0]     # 'P'
text[-1]    # 'n' (last character)

# Slicing
```

```
text[1:4]   # 'yth' (characters 1, 2, 3)
text[:3]    # 'Pyt' (first 3 characters)
text[2:]    # 'thon' (from position 2 to end)
text[::2]   # 'Pto' (every 2nd character)

# F-strings
name = "Alice"
age = 20
print(f"{name} is {age} years old")
```

2.5.3 Practice Questions

Practice 9: Given `word = "Programming"`, what do these expressions return?

```
a) word[3:7]
b) word[-4:]
c) word[::3]
```

Practice 10: What will this string method output?

```
sentence = "Hello World"
print(sentence.replace("World", "Python").upper())
```

2.6 6. Lists

2.6.1 What You Need to Know

- **Lists are mutable** (can be changed after creation)
- **List indexing and slicing** works like strings
- **List methods**: `.append()`, `.insert()`, `.remove()`, `.pop()`, `.copy()`
- **List cloning** vs **list referencing**

2.6.2 Key Points

```python
# Creating and modifying lists
fruits = ["apple", "banana", "cherry"]
fruits.append("date")               # Add to end
fruits.insert(1, "blueberry")       # Insert at position 1
fruits.remove("banana")             # Remove by value
last = fruits.pop()                 # Remove and return last item

# Cloning vs Referencing
list1 = [1, 2, 3]
list2 = list1           # Reference (same list!)
list3 = list1.copy()    # Clone (independent copy)

# List comprehensions
squares = [x**2 for x in range(5)]     # [0, 1, 4, 9, 16]
evens = [x for x in range(10) if x % 2 == 0]  # [0, 2, 4, 6, 8]
```

2.6.3 Practice Questions

Practice 11: What will this code output?

```python
numbers = [1, 2, 3]
numbers.append(4)
numbers.insert(0, 0)
print(numbers)
```

Practice 12: What's the difference between these two operations?

```python
original = [1, 2, 3]
copy_a = original
copy_b = original.copy()
original.append(4)
```

2.7 7. Tuples

2.7.1 What You Need to Know

- **Tuples are immutable** (cannot be changed after creation)

- Use **parentheses** instead of square brackets
- **Tuple unpacking**: `x, y, z = (1, 2, 3)`
- Good for coordinates, RGB values, or any fixed data

2.7.2 Key Points

```
# Creating tuples
point = (10, 20)
rgb = (255, 128, 0)
info = ("Alice", 20, "CS")

# Accessing elements (like lists)
x = point[0]        # 10
name = info[0]      # "Alice"

# Tuple unpacking
x, y = point        # x=10, y=20
name, age, major = info
```

2.7.3 Practice Questions

Practice 13: What will this code output?

```
data = (5, 10, 15, 20)
a, b, c, d = data
print(a + c)
```

2.8 8. Dictionaries

2.8.1 What You Need to Know

- **Key-value pairs**: `{"key": "value"}`
- **Accessing values**: `dict["key"]`
- **Adding/modifying**: `dict["new_key"] = "value"`
- **Dictionary methods**: `.keys()`, `.values()`, `.items()`

2.8.2 Key Points

```
# Creating and using dictionaries
student = {"name": "Bob", "age": 19, "grade": 85}

# Accessing and modifying
name = student["name"]          # "Bob"
student["age"] = 20             # Modify existing
student["gpa"] = 3.5            # Add new key-value pair

# Iterating through dictionaries
for key, value in student.items():
    print(f"{key}: {value}")
```

2.8.3 Practice Questions

Practice 14: What will this code output?

```
scores = {"Alice": 85, "Bob": 92, "Carol": 78}
scores["Alice"] = 87
print(len(scores))
print(scores["Bob"])
```

2.9 9. Sets

2.9.1 What You Need to Know

- **Sets contain unique elements** (no duplicates)
- **Set operations**: union (`|`), intersection (`&`), difference (`-`)
- Use **curly braces**: `{1, 2, 3}`

2.9.2 Key Points

```
# Creating sets
numbers = {3, 1, 4, 1, 5}       # Becomes {1, 3, 4, 5}
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

# Set operations
intersection = set1 & set2      # {3, 4}
union = set1 | set2             # {1, 2, 3, 4, 5, 6}
difference = set1 - set2        # {1, 2}
```

2.9.3 Practice Questions

Practice 15: What will this set operation return?

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
result = A & B
```

---

3 Algorithm Concepts

3.1 Exhaustive Enumeration

- **Systematic checking** of all possibilities
- Used for finding perfect squares, cubes, etc.
- **Limitation**: Only works for integer solutions

3.2 Newton's Method

- **Iterative approximation** technique
- Finds approximate roots of equations
- **Advantage**: Very fast convergence
- **Quadratic convergence**: Error roughly squares each iteration

---

4 Sample Practice Problems

4.1 Code Tracing Practice

Problem 1: What is the complete output?

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

Problem 2: What values do these variables have at the end?

```
x = 5
y = 10
x, y = y, x + 3
```

Problem 3: What will this nested condition print?

```
score = 85
if score >= 80:
    if score >= 90:
        print("Excellent")
    else:
        print("Good")
else:
    print("Needs improvement")
```

---

5 Common Mistakes to Avoid

5.1 Syntax Errors

- **Forgetting colons** after if, for, while, def statements
- **Inconsistent indentation** - Python is very strict about this
- **Using = instead of ==** for comparisons

5.2 Logic Errors

- **Off-by-one errors** in ranges and indexing
- **Confusing assignment and comparison** operators
- **Not handling edge cases** (empty lists, zero values, etc.)

5.3 String/List Confusion

- **Strings are immutable**, lists are mutable
- **String slicing** returns a string, not individual characters
- **List methods** like append() modify the original list

---

6 Final Exam Preparation Checklist

6.1 Before the Exam

☐ Review all code examples in this study guide
☐ Practice writing code by hand (no computer!)
☐ Work through all practice problems
☐ Understand the difference between similar concepts (lists vs tuples, = vs ==)
☐ Get a good night's sleep

6.2 During the Exam

☐ Read each question carefully
☐ Trace through code step by step
☐ Double-check your answers
☐ Manage your time (about 3 minutes per question)
☐ If stuck, move on and come back

6.3 Key Formulas to Remember

- **Range function**: `range(start, stop, step)`
- **String slicing**: `string[start:end:step]`
- **List comprehension**: `[expression for item in iterable if condition]`
- **F-string format**: `f"Text {variable} more text"`

Good luck on your midterm! Remember, understanding the concepts is more important than memorizing syntax. Focus on why things work the way they do, and you will be fine!