Tutorial I.II - Comparison Operators

Programming: Everyday Decision-Making Algorithms

Introduction

You're teaching a computer to make decisions. Just like we compare things in everyday life ("Is it raining?", "Do I have enough money?"), computers need ways to compare values and make choices. Let's look at this in the context of hiring decisions to help us how to make decisions in the context of optimal stopping problems.

Imagine you're helping a company make hiring decisions. You need to look at different candidates and decide if they meet certain requirements. To make these decisions with Python, we need to learn how to make comparisons!

1 - Basic Comparisons

When looking at job candidates, we often ask questions like:

- Does the candidate have enough experience?
- Is their salary request within our budget?
- Did they score well on the test?

These questions all require yes/no (`True/False`) answers, which is exactly what comparison operators give us in Python. Each comparison will result in either `True` or `False`.

In Python, we use special symbols to make these comparisons:

```
# Let's look at the current candidate compared to the best we've seen so far
current_candidate_score = 85
best_score_so_far = 82

# Is this the best candidate yet?
is_best_so_far = (current_candidate_score > best_score_so_far)
print(f"Is this our best candidate so far? {is_best_so_far}")

# Have we interviewed enough candidates?
candidates_seen = 15
total_candidates = 30
passed_threshold = (candidates_seen >= total_candidates * 0.37)
print(f"Have we passed our optimal stopping threshold? {passed_threshold}")
```

```
Is this our best candidate so far? True
Have we passed our optimal stopping threshold? True
```

> **Tip**
>
> When you make a comparison in Python, the result is always a boolean value (`True` or `False`). This is why we can store the result in variables like `is_best_so_far` - these variables will contain either `True` or `False`.

Here are all the comparison operators we can use:

Symbol	Meaning	Example
`==`	Equal to	`score == 100`
`!=`	Not equal to	`degree != "Computer Science"`
`<`	Less than	`salary < 80000`
`>`	Greater than	`experience > 5`
`<=`	Less than or equal to	`age <= 65`
`>=`	Greater than or equal to	`test_score >= 80`

Exercise 1.1 - Compare Test Scores

A candidate scored 95 on their test. The previous highest score was 88. Create a comparison to check if the new score is better and store the result in `compare_scores`.

```
# Creates the variables
new_score = 95
previous_best = 88
# YOUR CODE BELOW
```

```
# Test your answer
assert compare_scores == True, "The new score should be greater than the previous best"
print(f"Is new score better? {compare_scores}")
```

Exercise 1.2 - Compare Skills

Check if a candidate's skill matches what we need. Compare if the following two variables are different and store the result in `compare_skills`.

```
candidate_skill = "Finance"
required_skill = "Python"
# YOUR CODE BELOW
```

```
# Test your answer
assert compare_skills == False, "The candidate's skill should not match the required
 ↪  skill"
print("Good job! You correctly identified that the skills are different")
```

Exercise 1.3 - Compare Candidates

In the secretary problem, we need to compare candidates to the best we've seen. Check if the current candidate is better than our best so far and store the result in `is_better`.

```
current_rating = 92
best_so_far = 88
# YOUR CODE BELOW
```

```
# Test your answer
assert is_better == True, "The current candidate should be better than the best so far"
print("Good job! You correctly identified that the current candidate is better than the
 ↪  best so far")
```

2 - Combining Comparisons with Logical Operators

Often, we need to check multiple things at once. For example, a good candidate should:

- Have enough experience
- Be within our budget
- Pass the test

Python gives us three ways to combine comparisons:

- `and`: Both conditions must be true
- `or`: At least one condition must be true
- `not`: Makes true become false and false become true

Let's see some examples:

```
# Candidate information
experience = 5
test_score = 90
salary_ask = 75000

# Check multiple requirements at once
meets_experience = (experience >= 3)
good_test_score = (test_score >= 85)
affordable = (salary_ask <= 80000)

# All requirements must be met (using AND)
is_good_candidate = (meets_experience and good_test_score and affordable)
print(f"Should we hire this candidate? {is_good_candidate}")

# Alternative qualifications (using OR)
has_degree = True
has_certification = False
is_qualified = (has_degree or has_certification)
print(f"Is candidate qualified? {is_qualified}")
```

```
Should we hire this candidate? True
Is candidate qualified? True
```

> **!** Important
>
> Common Mistakes to Avoid:
> 1. Don't confuse = (assignment) with == (comparison)

2. Remember that `and` requires ALL conditions to be True
3. When combining multiple comparisons, use parentheses to make your logic clear Example: `(experience >= 3) and (test_score >= 85)`

Exercise 2.1 - Check Multiple Requirements

We have a candidate with:

- 4 years of experience
- Test score of 90

Check if they meet BOTH requirements:

- More than 3 years experience
- Test score above 85

Store the result in `meets_requirements`.

```
years = 4
score = 90
# YOUR CODE BELOW
```

```
# Test your answer
assert meets_requirements == True, "The candidate should meet both requirements"
print("Good job! You correctly identified that the candidate meets both requirements")
```

Exercise 2.2 - Alternative Requirements

A candidate is acceptable if they EITHER:

- Have a PhD
- OR have a high test score (above 95)

Store the result in `is_qualified`

```
has_phd = True
test_score = 92
# YOUR CODE BELOW
```

```
# Test your answer
assert is_qualified == True, "The candidate should be qualified"
print("Good job! You correctly identified that the candidate is qualified")
```

3 - Putting It All Together

Let's look at a real candidate and check all their qualifications:

```
# Secretary Problem Scenario
candidate_name = "Mika"
candidates_seen = 25
total_candidates = 50
current_score = 88
best_score_so_far = 85

# Our decision criteria
threshold_ratio = 0.37
passed_threshold = (candidates_seen >= total_candidates * threshold_ratio)
is_best_so_far = (current_score > best_score_so_far)

# Final decision - must pass threshold AND be best so far
should_hire = (passed_threshold and is_best_so_far)

print(f"Candidate: {candidate_name}")
print(f"Passed observation threshold: {passed_threshold}")
print(f"Best candidate so far: {is_best_so_far}")
print(f"Final decision - Should hire: {should_hire}")
```

```
Candidate: Mika
Passed observation threshold: True
Best candidate so far: True
Final decision - Should hire: True
```

Exercise 3.1 - Make a Hiring Decision

Look at the following candidate's information and decide if we should hire them:

Requirements:

- At least 5 years experience
- Test score above 85
- Salary request at most 90000
- Must have a degree

Create a single boolean expression that checks ALL requirements and store the result in `final_decision`.

```
# Candidate information
candidate_experience = 7
candidate_score = 89
```

```
candidate_salary = 92000
candidate_has_degree = True
# YOUR CODE BELOW
```

```
# Test your answer
assert final_decision == False, "The candidate should not meet all requirements"
print("Good job! You correctly identified that the candidate does not meet all
 ↪  requirements as his salary expectation is too high.")
```

Exercise 2.3 - Optimal Stopping Decision

We have:

- Seen 18 out of 40 candidates
- Current candidate score: 90
- Best score so far: 85

Check if we should hire this candidate (we should hire if we've seen at least 37% of candidates AND this candidate is better than the best so far).

Store the result in `make_offer`.

```
# Candidate information
candidates_seen = 18
total_candidates = 40
current_score = 90
best_score_so_far = 85
# YOUR CODE BELOW
```

```
# Test your answer
assert make_offer == True, "We should hire this candidate"
print("Good job! You correctly identified that we should hire this candidate")
```

Conclusion

Great job! You now know how to:

- Compare values using Python's comparison operators
- Combine comparisons using logical operators
- Make decisions based on multiple criteria

These skills are useful not just for hiring decisions, but for any situation where you need to make choices based on multiple conditions, especially in the context of optimal stopping problems!

Solutions

You will likely find solutions to most exercises online. However, we strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them next week. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.

Continue to the next tutorial to learn how to apply these skills to the optimal stopping problem!