SC1001: Introduction to Computational Thinking

Week 3: Boolean Logic and Branching

Jordan Boyd-Graber

Nanyang Technological University

Slides

Slides and updates will be posted at

`boydgraber.org/teaching/SC_1001/`

Overview

1. Predict the type and value of boolean expressions
2. Trace a program and predict what it prints
3. Evaluate an access-control condition step by step
4. Write a boolean rule for a smart home alarm
5. Trace named conditions in an access-control example
6. Follow an `if`/`elif`/`else` chain over a list
7. Optional: draw the control flow for an ATM menu

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`		
`1 // 2`		
`1 > 2`		
`True`		
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`		
`1 // 2`		
`1 > 2`		
`True`		
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`		
`1 > 2`		
`True`		
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`		
`True`		
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`		
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`		
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`	str	"True"
`3 < 3 and 3 / 0 > 0`		
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`	str	"True"
`3 < 3 and 3 / 0 > 0`	bool	False
`3 / 0 > 3 and 3 < 0`		
`3 > 0 or 3 / 0 != 1`		
`5 < 6 > 3`		

Because `3 < 3` is already `False`, Python short-circuits the `and`.

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`	str	"True"
`3 < 3 and 3 / 0 > 0`	bool	False
`3 / 0 > 3 and 3 < 0`	exception	ZeroDivisionError
`3 > 0 or 3 / 0 != 1` `5 < 6 > 3`		

Python must evaluate `3 / 0` first here, so it raises `ZeroDivisionError`.

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`	str	"True"
`3 < 3 and 3 / 0 > 0`	bool	False
`3 / 0 > 3 and 3 < 0`	exception	ZeroDivisionError
`3 > 0 or 3 / 0 != 1`	bool	True
`5 < 6 > 3`		

Because `3 > 0` is already `True`, Python short-circuits the `or`.

Discussion 1: Predict the Type and Value

For each expression, determine both its Python data type and its value.

Expression	Type	Value
`1 / 2`	float	0.5
`1 // 2`	int	0
`1 > 2`	bool	False
`True`	bool	True
`"True"`	str	"True"
`3 < 3 and 3 / 0 > 0`	bool	False
`3 / 0 > 3 and 3 < 0`	exception	ZeroDivisionError
`3 > 0 or 3 / 0 != 1`	bool	True
`5 < 6 > 3`	bool	True

`5 < 6 > 3` means `5 < 6 and 6 > 3`.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

Predict before running.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

Work through line by line.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

```
True
```

5 > 9 (False) and 10 < 8 (False), so left with: 20 > c > 9 means 20 > 11 and 11 > 9, so line1 prints True.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

```
True
10 3 2
```

Python evaluates the right-hand side first, then assigns all three variables together. What about `c // 3`, `floor(c / 3)`, `ceiling(c/3)`? (Assuming imported from `math`)

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

```
True
10 3 2
bigger
```

Since `a = 17`, the first `if` branch runs.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

```
True
10 3 2
bigger
small
```

This `print` is outside the first `if`, so it always runs.

Discussion 2: What Prints?

```
a, b, c = 9, 10, 11
print(5 > a or b < 8 or 20 > c > 9)  # L1

a, b, c = 4, -1, 11
c, b, a = a // 2, int(c / 3), b + c
print(a, b, c)                       # L2

a = 8 + 9
if a > 8:
    print("bigger")                  # L3
    print("small")                   # L4
if a < 9:
    print("bigger")                  # L5
print("small")                       # L6
```

Output

```
True
10 3 2
bigger
small
small
```

The second `if` is false, so `line5` prints nothing; then `line6` runs.

Discussion 3: What Is the Output?

```
is_admin = True
is_logged_in = True
has_permission = False

access_granted = is_logged_in and (is_admin or has_permission)

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Discussion 3: Evaluate the Boolean Expression

```
access_granted = is_logged_in and (is_admin or has_permission)
```

Discussion 3: Evaluate the Boolean Expression

$$\begin{aligned}\texttt{access_granted} &= \texttt{is_logged_in and (is_admin or has_permission)} \\ &= \texttt{True and (True or False)}\end{aligned}$$

Discussion 3: Evaluate the Boolean Expression

$$
\begin{aligned}
\texttt{access_granted} &= \texttt{is_logged_in and (is_admin or has_permission)} \\
&= \texttt{True and (True or False)} \\
&= \texttt{True and True}
\end{aligned}
$$

Discussion 3: Evaluate the Boolean Expression

$$
\begin{aligned}
\texttt{access_granted} &= \texttt{is_logged_in and (is_admin or has_permission)} \\
&= \texttt{True and (True or False)} \\
&= \texttt{True and True} \\
&= \texttt{True}
\end{aligned}
$$

Discussion 3: Evaluate the Boolean Expression

$$
\begin{aligned}
\texttt{access_granted} &= \texttt{is_logged_in and (is_admin or has_permission)} \\
&= \texttt{True and (True or False)} \\
&= \texttt{True and True} \\
&= \texttt{True}
\end{aligned}
$$

So the program prints `Access granted`.

Discussion 4: Write the Boolean Rule

Suppose we use these variables:

- `motion`: motion is detected
- `armed`: the alarm system is armed
- `door_open`: a door is open
- `window_open`: a window is open

Write a boolean expression that triggers the alarm exactly when there's motion, the system is armed, and either the window is open or the door is open.

Discussion 4: One Good Expression

```
alarm_triggered = motion and armed and (door_open or window_open)
```

Discussion 4: What If We Forget the Parentheses?

```
alarm_triggered = motion and armed and door_open or window_open
```

Python reads this as

(`motion and armed and door_open`) or `window_open`

So if `window_open` is `True`, the alarm goes off even without motion or an armed system.

Discussion 5: What Is the Output?

```
user_role = "administrator"
request_content = "delete record"

is_admin = (user_role.startswith("admin"))
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values Start with the given inputs.

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

`is_admin = True`

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

```
is_admin = True
is_manager = False
```

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

```
is_admin = True
is_manager = False
contains_delete = True
```

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

```
is_admin = True
is_manager = False
contains_delete = True
contains_record = True
```

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

```
is_admin = True
is_manager = False
contains_delete = True
contains_record = True

access_granted = False
```

$$(\texttt{True} \vee \texttt{False}) \wedge \texttt{True} \wedge \texttt{True} \;=\; \texttt{True} \wedge \texttt{True} \wedge \texttt{True} \;=\; \texttt{True}$$

Discussion 5: Trace the Conditions

```
user_role = "administrator"
request_content = "delete record"

is_admin = user_role.startswith("admin")
is_manager = (user_role == "manager")
contains_delete = ("delete" in request_content)
contains_record = ("record" in request_content)

access_granted = (is_admin or is_manager) and \
                 contains_delete and contains_record

if access_granted:
    print("Access granted.")
else:
    print("Access denied.")
```

Values

```
is_admin = True
is_manager = False
contains_delete = True
contains_record = True

access_granted = False
```

Output: `Access granted.`

List Membership vs. String Membership

```
sentence_string = "I am the very model of a modern major general"
words = sentence_string.split()

"mod" in sentence_string
"mod" in words
```

- `"mod" in sentence_string` is `True` because `"mod"` appears inside `"model"` and `"modern"`.
- `"mod" in words` is `False` because `"mod"` is not one of the list elements.

A Historical Python Note on Division

```
# Python 2 behavior
1 / 2      # 0
1 / 2.0    # 0.5

# Python 3 behavior
1 / 2      # 0.5
1 // 2     # 0
```

In Python 2, dividing two integers defaulted to integer division. Python 3 changed / to mean ordinary division.

Discussion 6: What Is the Output?

```
items = ["apple", "carrot", "durian", "tomato",
         "banana", "spinach", "orange", "broccoli"]
known_fruits = ["apple", "banana", "orange", "tomato"]
known_vegetables = ["carrot", "spinach", "broccoli", "tomato"]

num_fruit = num_veg = num_unknown = 0

for item in items:
    if item in known_fruits:
        num_fruit = num_fruit + 1
    elif item in known_vegetables:
        num_veg = num_veg + 1
    else:
        num_unknown = num_unknown + 1

print("Fruits:", num_fruit)
print("Vegetables:", num_veg)
print("Unknown:", num_unknown)
```

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1
banana	fruit	3	1	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1
banana	fruit	3	1	1
spinach	vegetable	3	2	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1
banana	fruit	3	1	1
spinach	vegetable	3	2	1
orange	fruit	4	2	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1
banana	fruit	3	1	1
spinach	vegetable	3	2	1
orange	fruit	4	2	1
broccoli	vegetable	4	3	1

Discussion 6: Walk Through the Items

Item	Branch taken	`num_fruit`	`num_veg`	`num_unknown`
apple	fruit	1	0	0
carrot	vegetable	1	1	0
durian	unknown	1	1	1
tomato	fruit	2	1	1
banana	fruit	3	1	1
spinach	vegetable	3	2	1
orange	fruit	4	2	1
broccoli	vegetable	4	3	1

Final output: `Fruits: 4`, `Vegetables: 3`, `Unknown: 1`.

Discussion 7 (Optional): ATM Menu Skeleton

```
balance = 1000.0

while True:
    print("""\nATM Menu:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit""")

    choice = int(input("Enter your choice (1-4): "))
```

How should we branch on `choice`?

Discussion 7 (Optional): Choice 1

```
if choice == 1:
    print("Current balance:", balance)
```

Discussion 7 (Optional): Choice 2

```
elif choice == 2:
    amount = float(input("Enter deposit amount: "))
    if amount > 0:
        balance = balance + amount
        print("New balance:", balance)
    else:
        print("Invalid deposit amount.")
```

Discussion 7 (Optional): Choice 3

```
elif choice == 3:
    amount = float(input("Enter withdrawal amount: "))
    if amount > 0 and amount <= balance:
        balance = balance - amount
        print("New balance:", balance)
    else:
        print("Invalid withdrawal amount.")
```

Discussion 7 (Optional): Choices 4 and Other

```
elif choice == 4:
    print("Thank you for using the ATM.")
    break
else:
    print("Invalid choice.")
```

Takeaways

- Boolean expressions can short-circuit, so order matters.
- Parentheses help communicate intent in complex logical conditions.
- Naming intermediate conditions makes program tracing easier.
- `if`/`elif`/`else` helps us separate cases cleanly.
- Branching code becomes easier to read when we trace one decision at a time.