

CN101

Lecture 4-5

Decision Structures and Boolean Logic

Topics

- The `if` Statement
- The `if-else` Statement
- Comparing Strings
- Nested Decision Structures and the `if-elif-else` Statement
- Logical Operators
- Boolean Variables

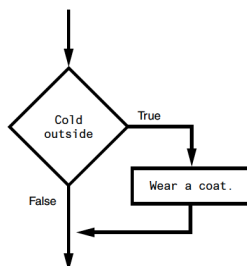
The `if` Statement

- Control structure: logical design that controls order in which set of statements execute
- Sequence structure: set of statements that execute in the order they appear
- Decision structure: specific action(s) performed only if a condition exists
 - Also known as selection structure

The `if` Statement (cont'd.)

- In flowchart, diamond represents true/false condition that must be tested
- Actions can be *conditionally executed*
 - Performed only when a condition is true
- Single alternative decision structure: provides only one alternative path of execution
 - If condition is not true, exit the structure

The `if` Statement (cont'd.)



The `if` Statement (cont'd.)

- Python syntax:

```
if condition:
    Statement
    Statement
```
- First line known as the `if` clause
 - Includes the keyword `if` followed by condition
 - The condition can be true or false
 - When the `if` statement executes, the condition is tested, and if it is true the block statements are executed. otherwise, block statements are skipped

Boolean Expressions and Relational Operators

- **Boolean expression:** expression tested by if statement to determine if it is true or false
 - Example: `a > b`
 - True if a is greater than b; False otherwise
- **Relational operator:** determines whether a specific relationship exists between two values
 - Example: greater than (`>`)

Boolean Expressions and Relational Operators (cont'd.)

- `>=` and `<=` operators test more than one relationship
 - It is enough for one of the relationships to exist for the expression to be true
- `==` operator determines whether the two operands are equal to one another
 - Do not confuse with assignment operator (`=`)
- `!=` operator determines whether the two operands are not equal

Boolean Expressions and Relational Operators (cont'd.)

Expression	Meaning
<code>x > y</code>	Is x greater than y?
<code>x < y</code>	Is x less than y?
<code>x >= y</code>	Is x greater than or equal to y?
<code>x <= y</code>	Is x less than or equal to y?
<code>x == y</code>	Is x equal to y?
<code>x != y</code>	Is x not equal to y?

Example

```

1 >>> x = 1 
2 >>> y = 0 
3 >>> y < x 
4 True
5 >>> x < y 
6 False
7 >>>

```

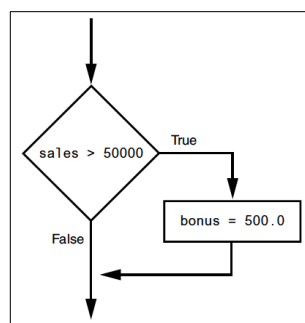
```

1 >>> x = 1 
2 >>> y = 0 
3 >>> z = 1 
4 >>> x >= y 
5 True
6 >>> x >= z 
7 True
8 >>> x <= z 
9 True
10 >>> x <= y 
11 False
12 >>>

```

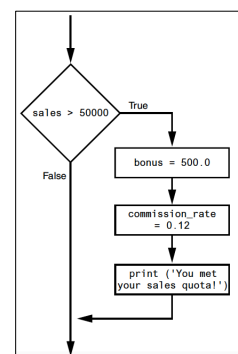
Putting It All Together

```
if sales > 50000:
    bonus = 500.0
```



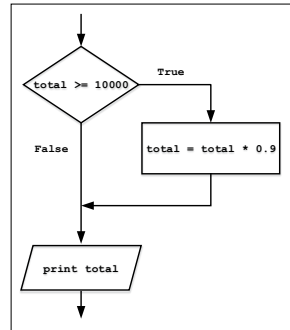
Putting It All Together

```
if sales > 50000:
    bonus = 500.0
    commission_rate = 0.12
    print('You met your sales quota!')
```



Putting It All Together

```
if total >= 10000:
    total = total * 0.9
print('Your total is', total)
```



Program 3-1 (test_average.py)

```
1 # This program gets three test scores and displays
2 # their average. It congratulates the user if the
3 # average is a high score.
4
5 # The HIGH_SCORE named constant holds the value that is
6 # considered a high score.
7 HIGH_SCORE = 95
8
9 # Get the three test scores.
10 test1 = int(input('Enter the score for test 1: '))
11 test2 = int(input('Enter the score for test 2: '))
12 test3 = int(input('Enter the score for test 3: '))
13
14 # Calculate the average test score.
15 average = (test1 + test2 + test3) / 3
16
17 # Print the average.
18 print('The average score is', average)
19
20 # If the average is a high score,
21 # congratulate the user.
22 if average >= HIGH_SCORE:
23     print('Congratulations!')
24     print('That is a great average!')
```

Program Output (with input shown in bold)

```
Enter the score for test 1: 82 Enter
Enter the score for test 2: 76 Enter
Enter the score for test 3: 91 Enter
The average score is 83.0
```

Program Output (with input shown in bold)

```
Enter the score for test 1: 93 Enter
Enter the score for test 2: 99 Enter
Enter the score for test 3: 96 Enter
The average score is 96.0
Congratulations!
That is a great average!
```

The if-else Statement

• Dual alternative decision structure: two possible paths of execution

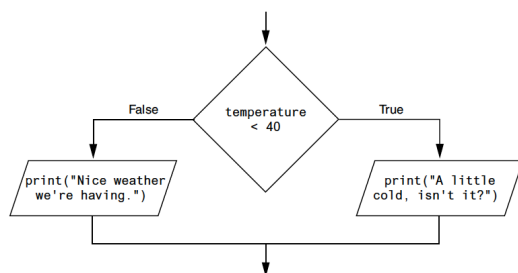
- One is taken if the condition is true, and the other if the condition is false

• Syntax: `if condition:`
 `statements`
 `else:`
 `other statements`

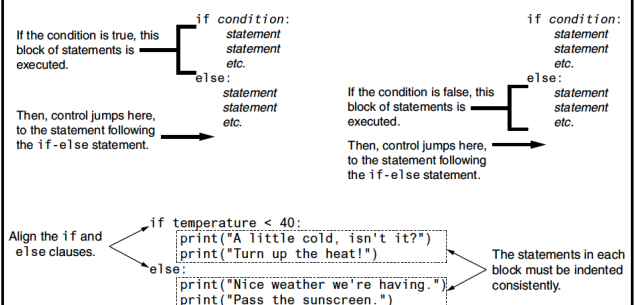
- if clause and else clause must be aligned
- Statements must be consistently indented

```
if condition:
    statement
    statement
    etc.
else:
    statement
    statement
    etc.
```

The if-else Statement (cont'd.)



The if-else Statement (cont'd.)



Program 3-2 (auto_repair_payroll.py)

```
1 # Named constants to represent the base hours and
2 # the overtime multiplier.
3 BASE_HOURS = 40 # Base hours per week
4 OT_MULTIPLIER = 1.5 # Overtime multiplier
5
6 # Get the hours worked and the hourly pay rate.
7 hours = float(input('Enter the number of hours worked: '))
8 pay_rate = float(input('Enter the hourly pay rate: '))
9
10 # Calculate and display the gross pay.
11 if hours > BASE_HOURS:
12     # Calculate the gross pay with overtime.
13     # First, get the number of overtime hours worked.
14     overtime_hours = hours - BASE_HOURS
15
16     # Calculate the amount of overtime pay.
17     overtime_pay = overtime_hours * pay_rate * OT_MULTIPLIER
18
19     # Calculate the gross pay.
20     gross_pay = BASE_HOURS * pay_rate + overtime_pay
21 else:
22     # Calculate the gross pay without overtime.
23     gross_pay = hours * pay_rate
24
25 # Display the gross pay.
26 print('The gross pay is $', format(gross_pay, '.,2f'), sep='')
```

Program Output (with input shown in bold)

Enter the number of hours worked: **40**
Enter the hourly pay rate: **20**
The gross pay is \$800.00.

Program Output (with input shown in bold)

Enter the number of hours worked: **50**
Enter the hourly pay rate: **20**
The gross pay is \$1,100.00.

Exercise

- Write an `if` statement that assigns 20 to the variable `y`, and assigns 40 to the variable `z` if the variable `x` is greater than 100.

Exercise

- Write a program that reads in a positive integer `n` from the user and then prints whether `n` is even or odd.
[Hint](#): An even number is a multiple of 2. Any multiple of 2 leaves a remainder of zero when divided by 2.

An example run of the program is shown below.

Enter a positive integer: 101
101 is odd

Comparing Strings

- Strings can be compared using the `==` and `!=` operators
- String comparisons are case sensitive
- Strings can be compared using `>`, `<`, `>=`, and `<=`
 - Compared character by character based on the ASCII values for each character
 - If shorter word is substring of longer word, longer word is greater than shorter word

ASCII Chart

TABLE 2: Standard ASCII code (in decimal)

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l
13	CR	29	GS	45	-	61	=	77	M	93]	109	m
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n
15	SI	31	US	47	/	63	?	79	O	95	_	111	o
												112	p
												113	q
												114	r
												115	s
												116	t
												117	u
												118	v
												119	w
												120	x
												121	y
												122	z
												123	{
												124	
												125	}
												126	~
												127	DEL

SP means space.

Comparing Strings (cont'd.)

```
name1 = 'Mary'
name2 = 'Mark'
if name1 > name2:
    print('Mary is greater than Mark')
else:
    print('Mary is not greater than Mark')
```

M a r y
77 97 114 121
↑ ↑ ↑ ↑
77 97 114 107
M a r k

Program 3-3 (password.py)

```
1 # This program compares two strings.
2 # Get a password from the user.
3 password = input('Enter the password: ')
4
5 # Determine whether the correct password
6 # was entered.
7 if password == 'prospero':
8     print('Password accepted.')
9 else:
10    print('Sorry, that is the wrong password.')
```

Program Output (with input shown in bold)

Enter the password: **ferdinand**
Sorry, that is the wrong password.

Program Output (with input shown in bold)

Enter the password: **prospero**
Password accepted.

Program 3-4 (sort_names.py)

```
1 # This program compares strings with the < operator.
2 # Get two names from the user.
3 name1 = input('Enter a name (last name first): ')
4 name2 = input('Enter another name (last name first): ')
5
6 # Display the names in alphabetical order.
7 print('Here are the names, listed alphabetically.')
8
9 if name1 < name2:
10    print(name1)
11    print(name2)
12 else:
13    print(name2)
14    print(name1)
```

Program Output (with input shown in bold)

Enter a name (last name first): **Jones, Richard**
Enter another name (last name first): **Costa, Joan**
Here are the names, listed alphabetically:
Costa, Joan
Jones, Richard

Nested Decision Structures and the if-elif-else Statement

- A decision structure can be nested inside another decision structure
 - Commonly needed in programs
 - Example:
 - Determine if someone qualifies for a loan, they must meet two conditions:
 - Must earn at least \$30,000/year
 - Must have been employed for at least two years
 - Check first condition, and if it is true, check second condition

Program 3-5 (loan_qualifier.py)

```
1 # This program determines whether a bank customer
2 # qualifies for a loan.
3
4 MIN_SALARY = 30000.0 # The minimum annual salary
5 MIN_YEARS = 2 # The minimum years on the job
6
7 # Get the customer's annual salary.
8 salary = float(input('Enter your annual salary: '))
9
10 # Get the number of years on the current job.
11 years_on_job = int(input('Enter the number of ' +
12     'years employed: '))
13
14 # Determine whether the customer qualifies.
15 if salary >= MIN_SALARY:
16     if years_on_job >= MIN_YEARS:
17         print('You qualify for the loan.')
18     else:
19         print('You must have been employed',
20             'for at least', MIN_YEARS,
21             'years to qualify.')
22 else:
23     print('You must earn at least $',
24         format(MIN_SALARY, ',.2f'),
25         'per year to qualify.', sep='')
```

Program Output (with input shown in bold)

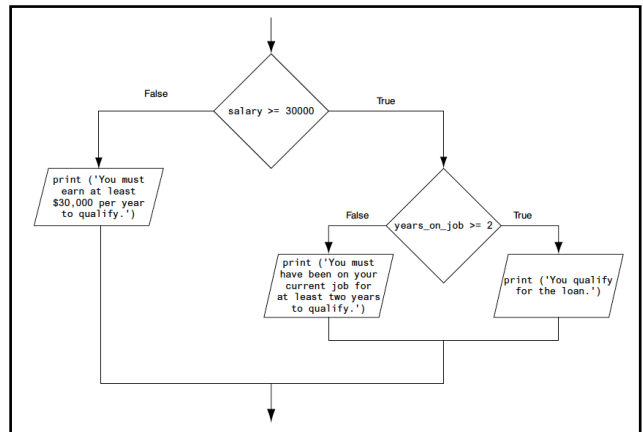
Enter your annual salary: **35000**
Enter the number of years employed: **1**
You must have been employed for at least 2 years to qualify.

Program Output (with input shown in bold)

Enter your annual salary: **25000**
Enter the number of years employed: **5**
You must earn at least \$30,000.00 per year to qualify.

Program Output (with input shown in bold)

Enter your annual salary: **35000**
Enter the number of years employed: **5**
You qualify for the loan.



Nested Decision Structures and the if-elif-else Statement (cont'd.)

- Important to use proper indentation in a nested decision structure
 - Important for Python interpreter
 - Makes code more readable for programmer
- Rules for writing nested if statements:
 - else clause should align with matching if clause
 - Statements in each block must be consistently indented

This if and else go together. → if salary >= MIN_SALARY:
This if and else go together. → if years_on_job >= MIN_YEARS:
→ print('You qualify for the loan.')
→ else:
→ print('You must have been employed 'for at least', MIN_YEARS, 'years to qualify.')
→ else:
→ print('You must earn at least \$', format(MIN_SALARY, ',.2f'), ' per year to qualify.', sep='')

The if-elif-else Statement

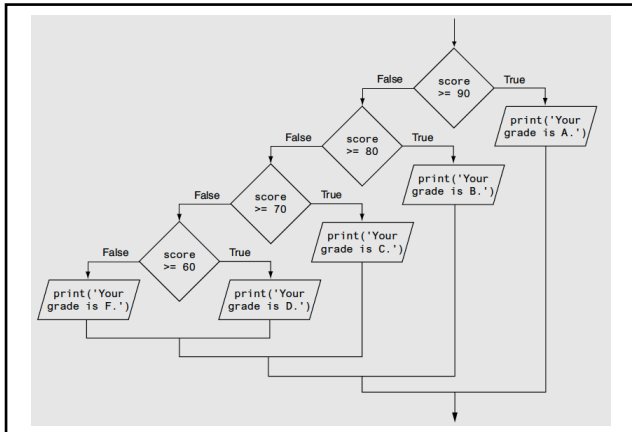
- if-elif-else statement: special version of a decision structure
 - Makes logic of nested decision structures simpler to write
 - Can include multiple elif statements
- Syntax:

```
if condition_1:
    statement(s)
elif condition_2:
    statement(s)
elif condition_3:
    statement(s)
else:
    statement(s)
```

} Insert as many elif clauses as necessary.

The if-elif-else Statement (cont'd.)

- Alignment used with if-elif-else statement:
 - if, elif, and else clauses are all aligned
 - Conditionally executed blocks are consistently indented
- if-elif-else statement is never required, but logic easier to follow
 - Can be accomplished by nested if-else
 - Code can become complex, and indentation can cause problematic long lines



Program 3-6 (grader.py)

```

1 # This program gets a numeric test score from the
2 # user and displays the corresponding letter grade.
3
4 # Named constants to represent the grade thresholds
5 A_SCORE = 90
6 B_SCORE = 80
7 C_SCORE = 70
8 D_SCORE = 60
9
10 # Get a test score from the user.
11 score = int(input('Enter your test score: '))
12
13 # Determine the grade.
14 if score >= A_SCORE:
15     print('Your grade is A.')
16 else:
17     if score >= B_SCORE:
18         print('Your grade is B.')
19     else:
20         if score >= C_SCORE:
21             print('Your grade is C.')
22         else:
23             if score >= D_SCORE:
24                 print('Your grade is D.')
25             else:
26                 print('Your grade is F.')
  
```

Program Output (with input shown in bold)
Enter your test score: **78** (Enter)
Your grade is C.

Program Output (with input shown in bold)
Enter your test score: **84** (Enter)
Your grade is B.

```

if score >= A_SCORE:
    print('Your grade is A.')
elif score >= B_SCORE:
    print('Your grade is B.')
elif score >= C_SCORE:
    print('Your grade is C.')
elif score >= D_SCORE:
    print('Your grade is D.')
else:
    print('Your grade is F.')
  
```

Logical Operators

- **Logical operators:** operators that can be used to create complex Boolean expressions
 - **and operator** and **or operator:** binary operators, connect two Boolean expressions into a compound Boolean expression
 - **not operator:** unary operator, reverses the truth of its Boolean operand

The and Operator

- Takes two Boolean expressions as operands
 - Creates compound Boolean expression that is true only when both sub expressions are true
 - Can be used to simplify nested decision structures
- Truth table for the **and** operator

Expression	Value of the Expression
False and False	False
False and True	False
True and False	False
True and True	True

The or Operator

- Takes two Boolean expressions as operands
 - Creates compound Boolean expression that is true when either of the sub expressions is true
 - Can be used to simplify nested decision structures
- Truth table for the **or** operator

Expression	Value of the Expression
False or False	False
False or True	True
True or False	True
True or True	True

Short-Circuit Evaluation

- **Short circuit evaluation:** deciding the value of a compound Boolean expression after evaluating only one sub expression
 - Performed by the **or** and **and** operators
 - For **or** operator: If left operand is true, compound expression is true. Otherwise, evaluate right operand
 - For **and** operator: If left operand is false, compound expression is false. Otherwise, evaluate right operand

The not Operator

- Takes one Boolean expressions as operand and reverses its logical value
 - Sometimes it may be necessary to place parentheses around an expression to clarify to what you are applying the not operator
- Truth table for the not operator

Expression	Value of the Expression
not True	False
not False	True

Checking Numeric Ranges with Logical Operators

- To determine whether a numeric value is within a specific range of values, use **and**
 - Example: `x >= 10 and x <= 20`
 - Can also be written as: `10 <= x <= 20`

```
if x >= 10 and x <= 20:
    print('The value is in the acceptable range.')
```

Checking Numeric Ranges with Logical Operators (cont'd)

- To determine whether a numeric value is outside of a specific range of values, use **or**
 - Example: `x < 10 or x > 20`

```
if x < 10 or x > 20:
    print('The value is outside the acceptable range.')
```

Program 3-7 (loan_qualifier2.py)

```
1 # This program determines whether a bank customer
2 # qualifies for a loan.
3
4 MIN_SALARY = 30000.0 # The minimum annual salary
5 MIN_YEARS = 2        # The minimum years on the job
6
7 # Get the customer's annual salary.
8 salary = float(input('Enter your annual salary: '))
9
10 # Get the number of years on the current job.
11 years_on_job = int(input('Enter the number of ' +
12                          'years employed: '))
13
14 # Determine whether the customer qualifies.
15 if salary >= MIN_SALARY and years_on_job >= MIN_YEARS:
16     print('You qualify for the loan.')
17 else:
18     print('You do not qualify for this loan.')
```

Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 1 Enter
You do not qualify for this loan.
```

Program Output (with input shown in bold)

```
Enter your annual salary: 25000 Enter
Enter the number of years employed: 5 Enter
You do not qualify for this loan.
```

Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 5 Enter
You qualify for the loan.
```

Boolean Variables

- **Boolean variable**: references one of two values, **True** or **False**
 - Represented by `bool` data type
- Commonly used as flags
 - **Flag**: variable that signals when some condition exists in a program
 - Flag set to **False** → condition does not exist
 - Flag set to **True** → condition exists

Boolean Variables (cont'd)

- For example, the following code determines whether the quota of \$50,000 of a salesperson has been met:

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
```

Boolean Variables (cont'd)

- The bool variable `sales_quota_met` can be used as a flag to indicate whether the sales quota has been met. We can test the flag in the following way:

```
if sales_quota_met == True:
    print('You have met your sales quota!')
```

or

```
if sales_quota_met:
    print('You have met your sales quota!')
```

TABLE 1: Operator precedence and associativity

Level	Operators	Description	Associativity
1	<code>(expressions...)</code> , <code>[expressions...]</code> , <code>{key:value...}</code> , <code>{expressions...}</code>	binding or tuple display, list display, dictionary display, set display	left to right
2	<code>x[index]</code> , <code>x[index:index]</code> , <code>x(arguments...)</code> , <code>x.attribute</code>	subscripting, slicing, call, attribute reference	left to right
3	<code>await x</code>	await expression	left to right
4	<code>**</code>	exponentiation	right-to-left
5	<code>+x</code> , <code>-x</code> , <code>~x</code>	positive, negative, bitwise NOT	left-to-right
6	<code>*</code> , <code>@</code> , <code>/</code> , <code>//</code> , <code>%</code>	multiplication, matrix multiplication, division, floor division, remainder	left-to-right
7	<code>+x</code> , <code>-x</code>	addition, subtraction	left to right
8	<code><<</code> , <code>>></code>	shifts	left to right
9	<code>&</code>	bitwise AND	left to right
10	<code>^</code>	bitwise XOR	left to right
11	<code> </code>	bitwise OR	left to right
12	<code>in</code> , <code>not in</code> , <code>is</code> , <code>is not</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>!=</code> , <code>==</code>	comparisons, including membership tests and identity tests	left to right
13	<code>not x</code>	boolean NOT	left to right
14	<code>and</code>	boolean AND	left to right
15	<code>or</code>	boolean OR	left to right
16	<code>if - else</code>	conditional expression	left to right
17	<code>lambda</code>	lambda expression	left to right

Operators are shown in decreasing order of precedence from level 1 to level 17.

Summary

- This chapter covered:
 - Decision structures, including:
 - Single alternative decision structures
 - Dual alternative decision structures
 - Nested decision structures
 - Relational operators and logical operators as used in creating Boolean expressions
 - String comparison as used in creating Boolean expressions
 - Boolean variables

Exercise

- Write a program that asks the user to enter 3 integers. The program should display the maximum number.

An example run of the program is shown below.

```
Enter integer #1: 10
Enter integer #2: 35
Enter integer #3: 9
The maximum number is 35
```

Exercise

A software company sells a package that retails for \$99. Quantity discounts are given according to the following table:

Quantity	Discount
10–19	10%
20–49	20%
50–99	30%
100 or more	40%

Write a program that asks the user to enter the number of packages purchased. The program should then display the amount of the discount (if any) and the total amount of the purchase after the discount.