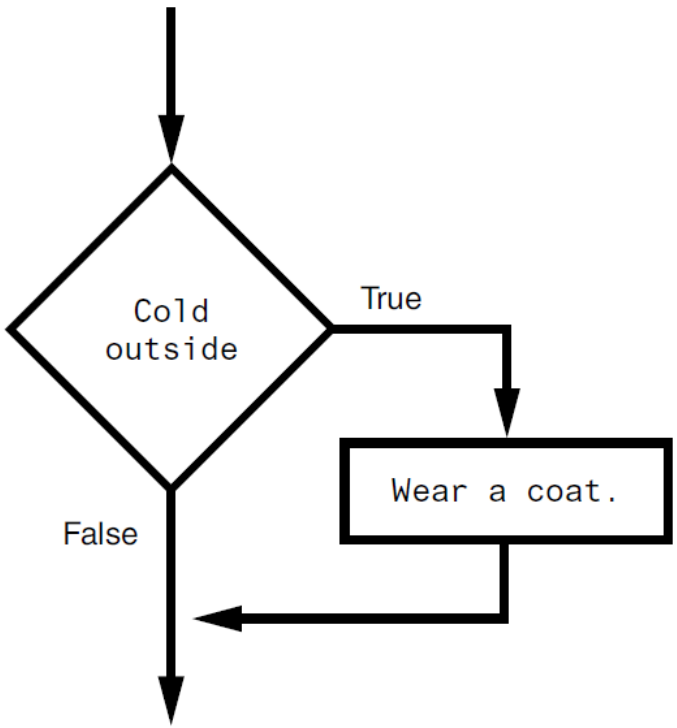
Decision Structures and Boolean Logic

陳建良

The if Statement

- A *control structure* is a logical design that controls the order in which a set of statements execute.
- So far, we have used only the simplest type of control structure: the sequence structure.
- A *sequence structure* is a set of statements that execute in the order in which they appear.
- For example,
  - name = input('What is your name? ')
  - age = int(input('What is your age? '))
  - print('Here is the data you entered:')
  - print('Name:', name)
  - print('Age:', age)

- Programs like this require a different type of control structure: one that can execute a set of statements only under certain circumstances.
- This can be accomplished with a ***decision structure***.
- Decision structures are also known as ***selection structures***.
- In a decision structure's simplest form, a specific action is performed only if a certain condition exists.
- If the condition does not exist, the action is not performed.



- The action is ***conditionally executed*** because it is performed only when a certain condition is true.
- Programmers call the type of decision structure ***a single alternative decision structure***.

- This is because it provides only one alternative path of execution.
- If the condition in the diamond symbol is true, we take the alternative path. Otherwise, we exit the structure.
- In Python, we use the **if** statement to write a single alternative decision structure.
- Here is the general format of the if statement:

the "if' keyword begins a selection statement

condition to be tested

```
if condition:
    statement
    statement
    statement
```

colon denotes end of condition

statements to be executed if condition is true

"block" of execution must be consistently indented

Boolean Expressions and Relational Operators

- The **if** statement tests an expression to determine whether it is true or false.
- The expressions that are tested by the if statement are called ***Boolean expressions***.
- The Boolean expression that is tested by an **if** statement is formed with a relational operator.
- A ***relational operator*** determines whether a specific relationship exists between two values.
- ALL Boolean expressions boil down to "**True**" or "**False**".
- Programmers often say that the expression "evaluates" to "True" or "False"

- For example, the greater than operator (>) determines whether one value is greater than another.
- The equal to operator (==) determines whether two values are equal.

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

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

Writing Boolean Expressions

```
pen = 10
sword = 7

if pen > sword:                    # pen > sword

 print ('the pen is               # 10 > 7
 mightier than the
 sword!')                          # True
```

Let Us Evaluate!

```
# given these variables

a = 99
b = 7
c = -5
d = 92
```

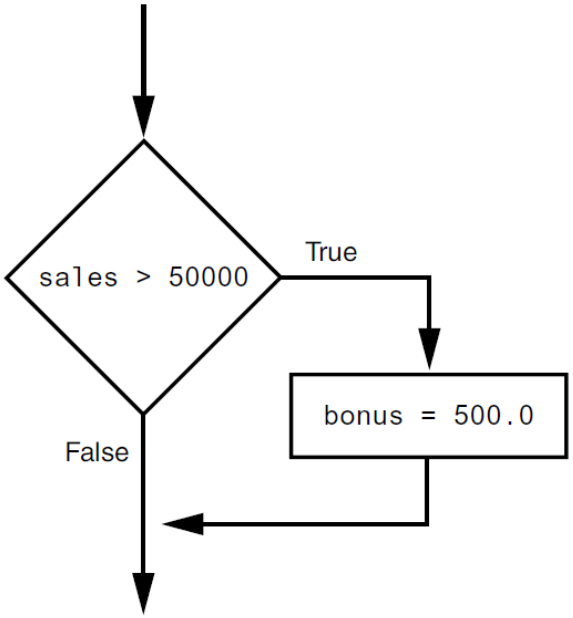
```
# evaluate these expressions

a > b
b < c
b >= c
c <= d
a == b + d
d <= a + c
c != b
```

PUTTING IT ALL TOGETHER

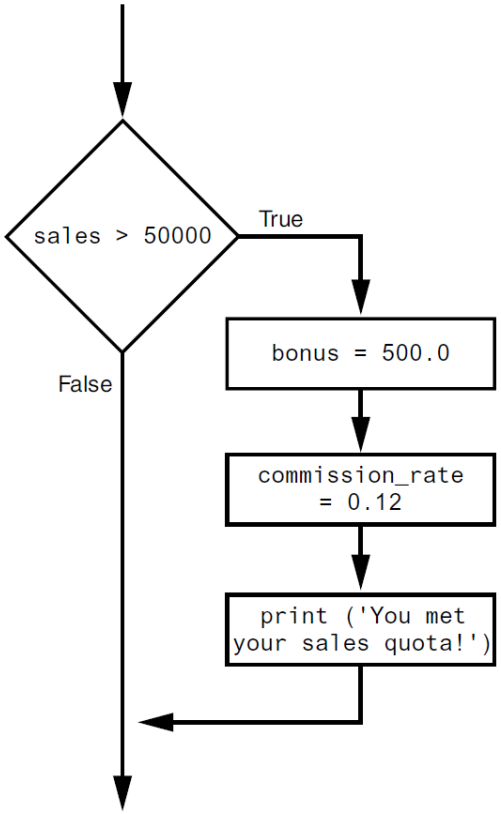
- Let's look at the following example of the if statement:

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



- The following example conditionally executes a block containing three statements.

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



- Any relational operator can be used in a decision block
  - Example: `if balance == 0`
  - Example: `if payment != balance`
- It is possible to have **a block inside another block**
  - Example: `if` statement inside a function
  - Statements in inner block must be indented with respect to the outer block

Boolean Operator Tips

- Don't confuse "==" with "="
  - "=" is used for assigning values to variables.
  - "==" is used for testing to see if two values are identical.
- Use "!=" if you want to test if two values are different.
- The "<=" and ">=" operators test for more than one relationship.
  - "<=" tests to see if a value is less than OR equal to another
  - ">=" tests to see if a value is greater than OR equal to another

- Kathryn teaches a science class and her students are required to take three tests. She wants to write a program that her students can use to calculate their average test score. She also wants the program to congratulate the student enthusiastically if the average is greater than 95. Here is the algorithm in pseudocode:

  Get the first test score

  Get the second test score

  Get the third test score

  Calculate the average

  Display the average

  If the average is greater than 95:

  Congratulate the user

Program Output (with input shown in bold)

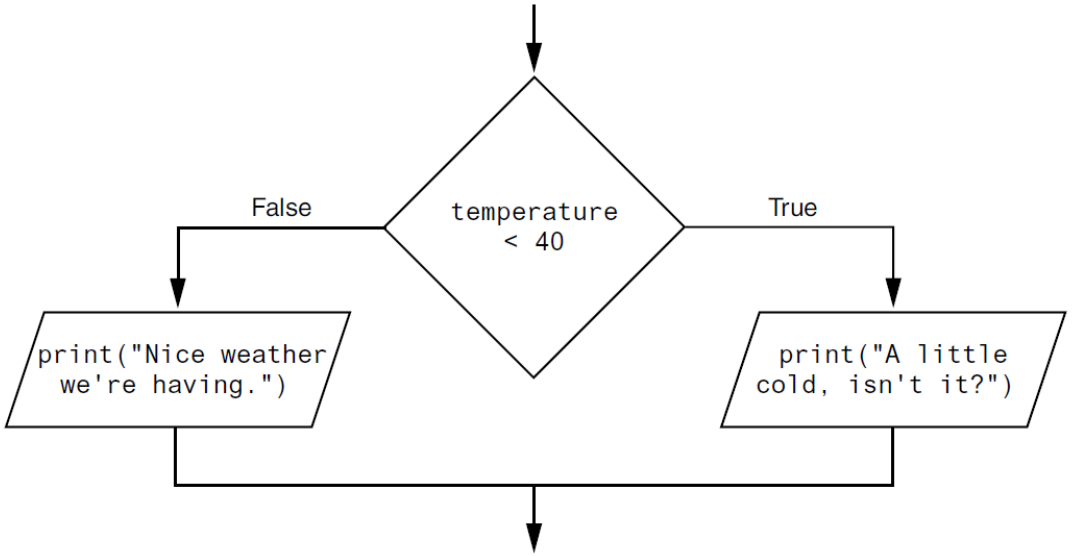
```
Enter the score for test 1: 82 Enter
Enter the score for test 2: 76 Enter
Enter the score for test 3: 91 Enter
The average score is 83.0
```

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!
```

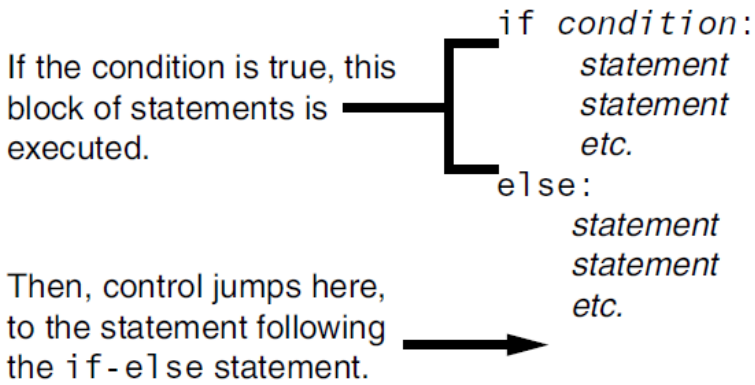
The if-else Statement

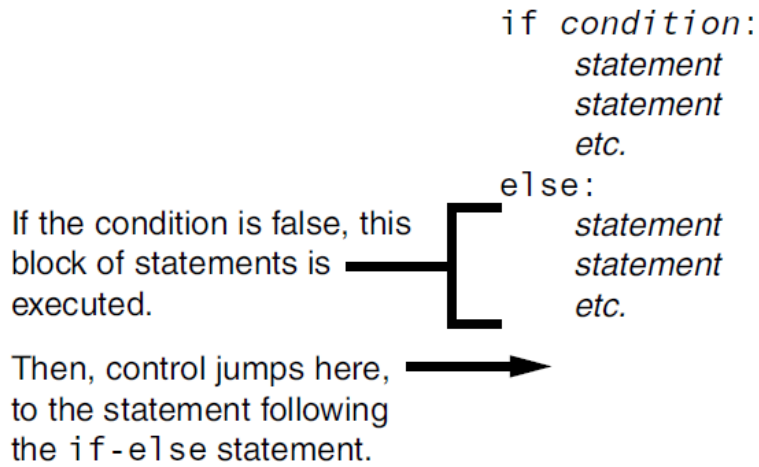
- An if-else statement will execute one block of statements if its condition is true, or another block if its condition is false.
- the *dual* ***alternative decision structure***, which has two possible paths of execution—one path is taken if a condition is true, and the other path is taken if the condition is false.



- **Syntax:**

```
if condition:
    statements
else:
    other statements
```





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

Align the `if` and `else` clauses.

```
if temperature < 40:
    print("A little cold, isn't it?")
    print("Turn up the heat!")
else:
    print("Nice weather we're having.")
    print("Pass the sunscreen.")
```

The statements in each block must be indented consistently.

- Chris owns an auto repair business and has several employees. If any employee works over 40 hours in a week, he pays them 1.5 times their regular hourly pay rate for all hours over 40. He has asked you to design a simple payroll program that calculates an employee's gross pay, including any overtime wages. You design the following algorithm:

Get the number of hours worked.

Get the hourly pay rate.

If the employee worked more than 40 hours:

Calculate and display the gross pay with overtime.

Else:

Calculate and display the gross pay as usual.

Program Output (with input shown in bold)

```
Enter the number of hours worked: 40 [Enter]
Enter the hourly pay rate: 20 [Enter]
The gross pay is $800.00.
```

Program Output (with input shown in bold)

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

Comparing Strings

- Python allows you to compare strings. This allows you to create decision structures that test the value of a string.

```
name1 = 'Mary'
name2 = 'Mark'
if name1 == name2:
  print('The names are the same.')
else:
  print('The names are NOT the same.')
```

- The program prompts the user to enter a password, then determines whether the string entered is equal to 'prospero'.

Program Output (with input shown in bold)

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

Program Output (with input shown in bold)

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

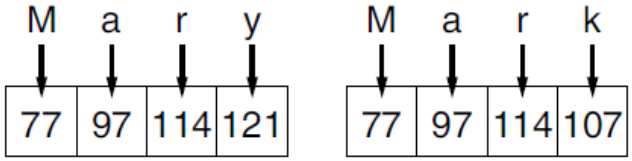
Other String Comparisons

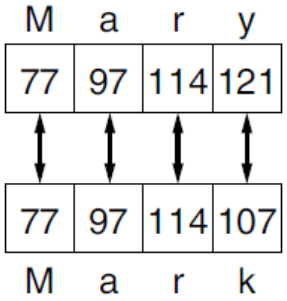
- Computers do not actually store characters, such as A, B, C, and so on, in memory.
- They store numeric codes that represent the characters.
- ASCII (the American Standard Code for Information Interchange) is a commonly used character coding system.
  - The uppercase characters A through Z are represented by the numbers 65 through 90.
  - The lowercase characters a through z are represented by the numbers 97 through 122.
  - When the digits 0 through 9 are stored in memory as characters, they are represented by the numbers 48 through 57. (For example, the string 'abc123' would be stored in memory as the codes 97, 98, 99, 49, 50, and 51.)
  - A blank space is represented by the number 32.

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

- When you use relational operators to compare these strings, the strings are compared **character-by-character**. For example,

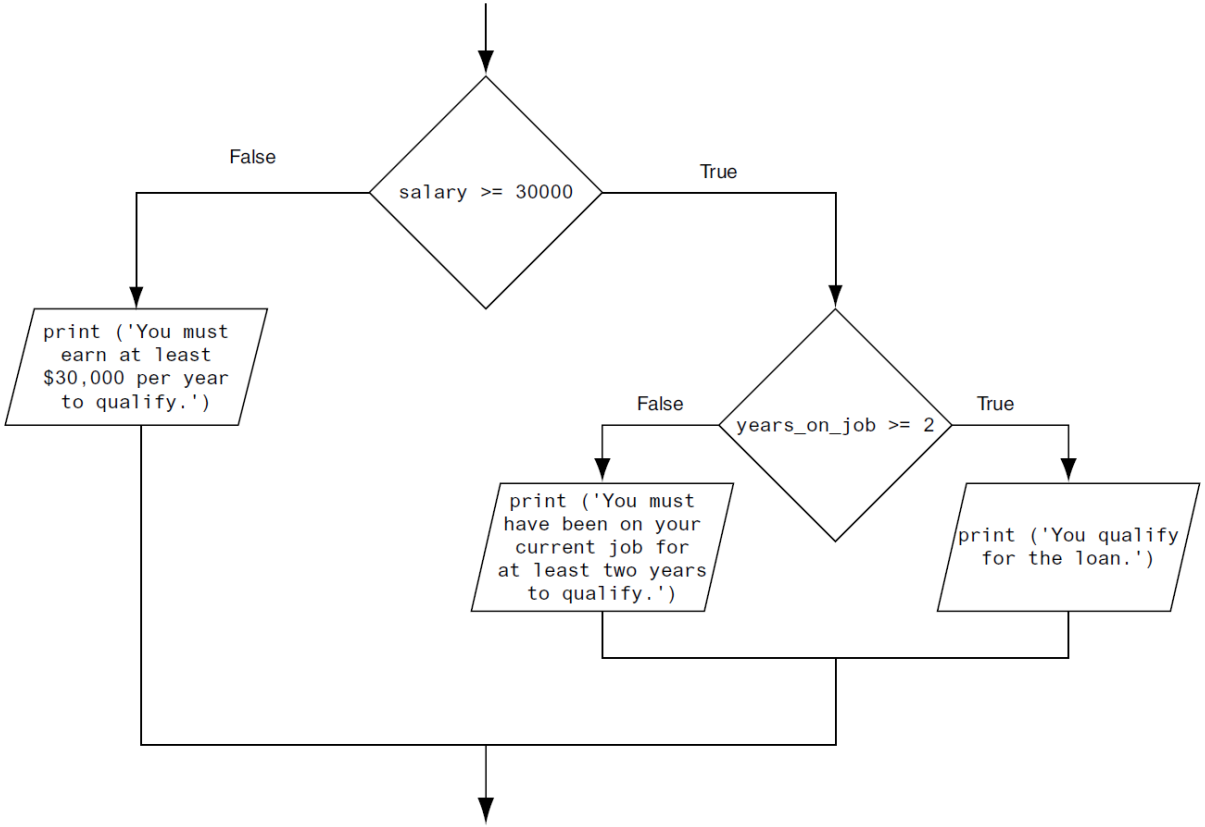
```
name1 = 'Mary'
name2 = 'Mark'
if name1 > name2:
    print('Mary is greater than Mark')
else:
    print('Mary is not greater than Mark')
```





Nest Decision Structures

- Python allows you to "nest" decision structures inside one another, allowing you to evaluate additional conditions.



```
# This program determines whether a bank customer
# qualifies for a loan.

MIN_SALARY = 30000.0 # The minimum annual salary
MIN_YEARS = 2        # The minimum years on the job

# Get the customer's annual salary.
salary = float(input('Enter your annual salary: '))

# Get the number of years on the current job.
years_on_job = int(input('Enter the number of' +
                         'years employed: '))

# Determine whether the customer qualifies.
if salary >= MIN_SALARY:
    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='')
```

Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 1 Enter
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
Enter the number of years employed: 5 Enter
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
Enter the number of years employed: 5 Enter
You qualify for the loan.
```

This if and else go together.

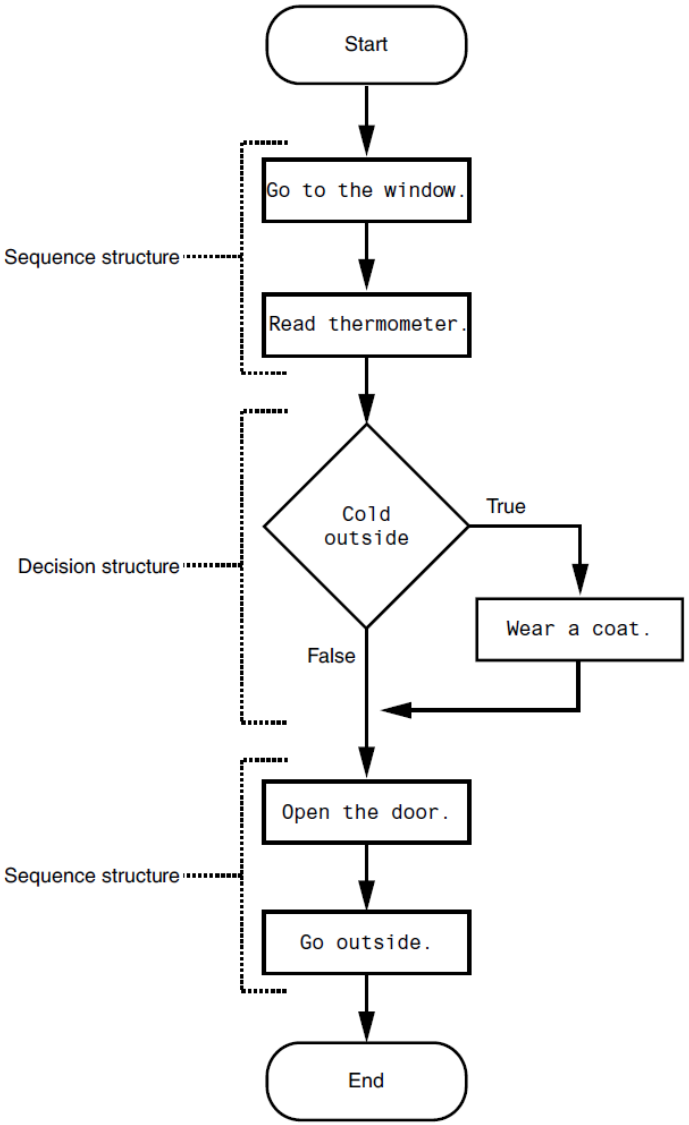
This if and else go together.

```
if salary >= MIN_SALARY:
    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='')
```

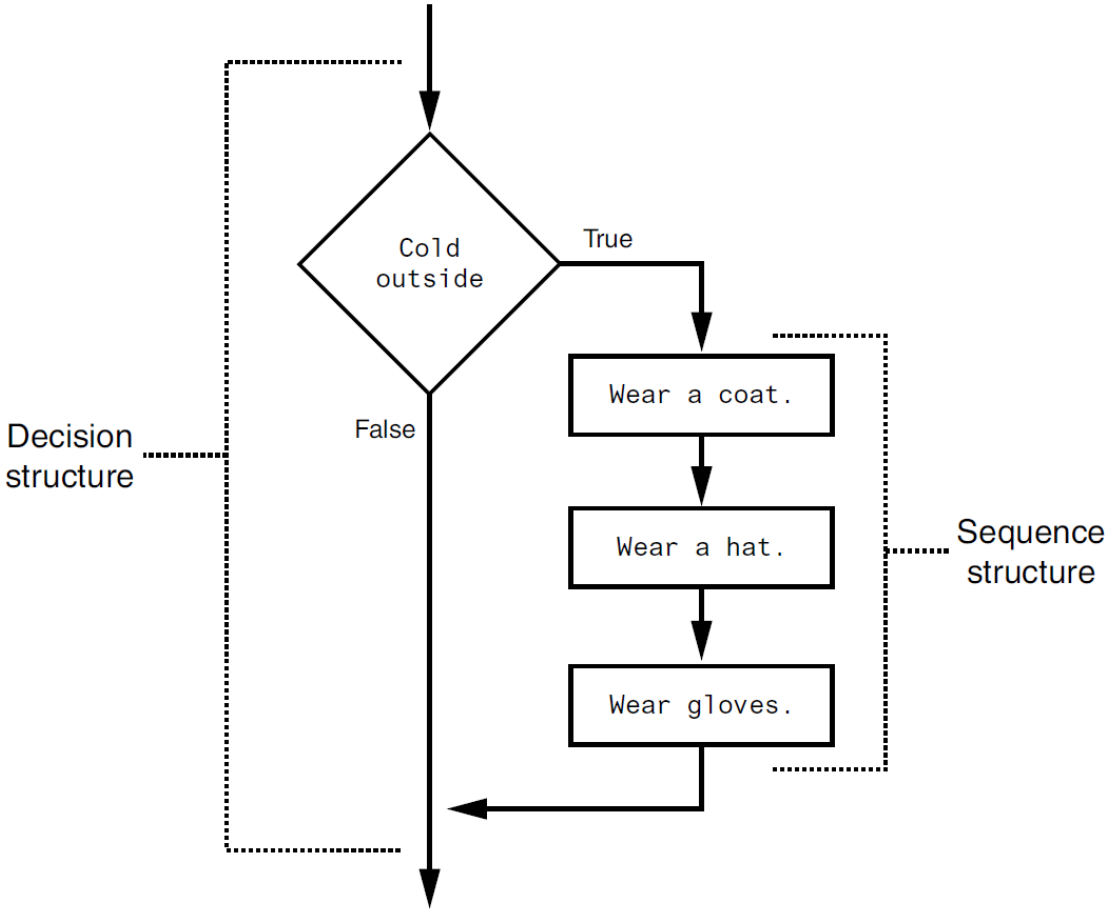
Nested blocks

```
if salary >= MIN_SALARY:
    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='')
```

Combining sequence structures with a decision structure



A sequence structure nested inside a decision structure



Testing a Series of Conditions -- Multiple Nested Decision Structures

- Dr. Suarez teaches a literature class and uses the following 10-point grading scale for all of his exams:

Test Score	Grade
90 and above	A
80–89	B
70–79	C
60–69	D
Below 60	F

- He has asked you to write a program that will allow a student to enter a test score and then display the grade for that score. Here is the algorithm that you will use:

1. Ask the user to enter a test score.
2. Determine the grade in the following manner:

If the score is greater than or equal to 90, then the grade is A.

Else, if the score is greater than or equal to 80, then the grade is B.

Else, if the score is greater than or equal to 70, then the grade is C.

Else, if the score is greater than or equal to 60, then the grade is D.

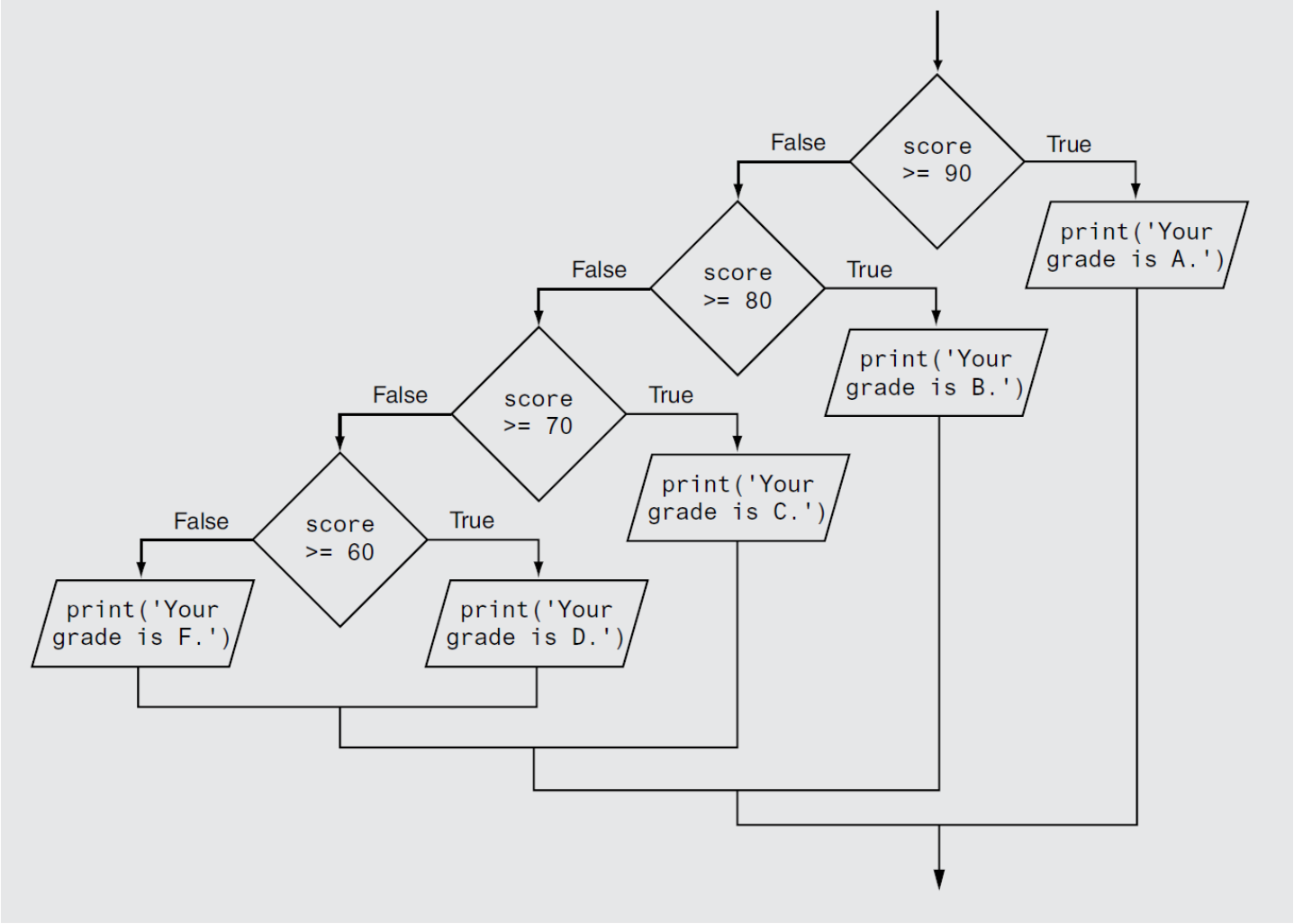
Else, the grade is F.

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.
```



The if-elif-else Statement

- Python provides a special version of the decision structure known as the if-elif-else statement, which makes this type of logic simpler to write.

if *condition_1:*

statement

statement

etc.

elif *condition_2:*

statement

statement

etc.

else:

statement

statement

etc.

```
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

- Python provides a set of operators known as ***logical operators***, which you can use to create complex Boolean expressions.
- There are three main logical operators that we use regularly in programming.
  - and
  - or
  - not

```
x = 10
y = 5
a = 20
b = 25

if x > y and a < b:
    print ('yes!')
else:
    print ('no!')
```

- several compound Boolean expressions that use logical operators.

Expression	Meaning
`x > y and a < b`	Is x greater than y AND is a less than b?
`x == y or x == z`	Is x equal to y OR is x equal to z?
`not (x > y)`	Is the expression x > y NOT true?

The and Operator

Expression	Value of the Expression
true and false	false
false and true	false
false and false	false
true and true	true

- The following is an example of an if statement that uses the and operator:

```
if temperature < 20 and minutes > 12:
    print('The temperature is in the danger zone.')
```

The or Operator

Expression	Value of the Expression
true or false	true
false or true	true
false or false	false
true or true	true

- An example of an if statement that uses the or operator:

```
if temperature < 20 or temperature > 100:
    print('The temperature is too extreme')
```

The not Operator

Expression	Value of the Expression
not true	false
not false	true

- An if statement using the not operator.

```
if not(temperature > 100):
    print('This is below the maximum temperature.')
```

Checking Numeric Ranges with Logical Operators

- if statement checks the value in x to determine whether it is in the range of 20 through 40.

```
if x >= 20 and x <= 40:
    print('The value is in the acceptable range.')
```

Boolean Variables

- A Boolean variable can reference one of two values: True or False.
- Boolean variables are commonly used as flags, which indicate whether specific conditions exist.
- When the flag variable is set to False, it indicates the condition does not exist.
- When the flag variable is set to True, it means the condition does exist.

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
if sales_quota_met:
    print('You have met your sales quota!')
```

Turtle Graphics: Determining the State of the Turtle

- Determining the Turtle's Location
- you can use the **turtle.xcor()** and **turtle.ycor()** functions to get the turtle's current *X* and *Y* coordinates.

```
if turtle.xcor() > 249 or turtle.ycor() > 349:
    turtle.goto(0, 0)
```

- Determining the Turtle's Heading

```
if turtle.heading() >= 90 and turtle.heading() <= 270:
    turtle.setheading(180)
```

- Determining Whether the Pen Is Down

```
if turtle.isdown():
    turtle.penup()
```

- Determining Whether the Turtle Is Visible

```
if turtle.isvisible():
    turtle.hideturtle()
```

- Determining the Current Colors

```
if turtle.pencolor() == 'red':
    turtle.pencolor('blue')
if turtle.bgcolor() == 'white':
    turtle.fillcolor('gray')
```

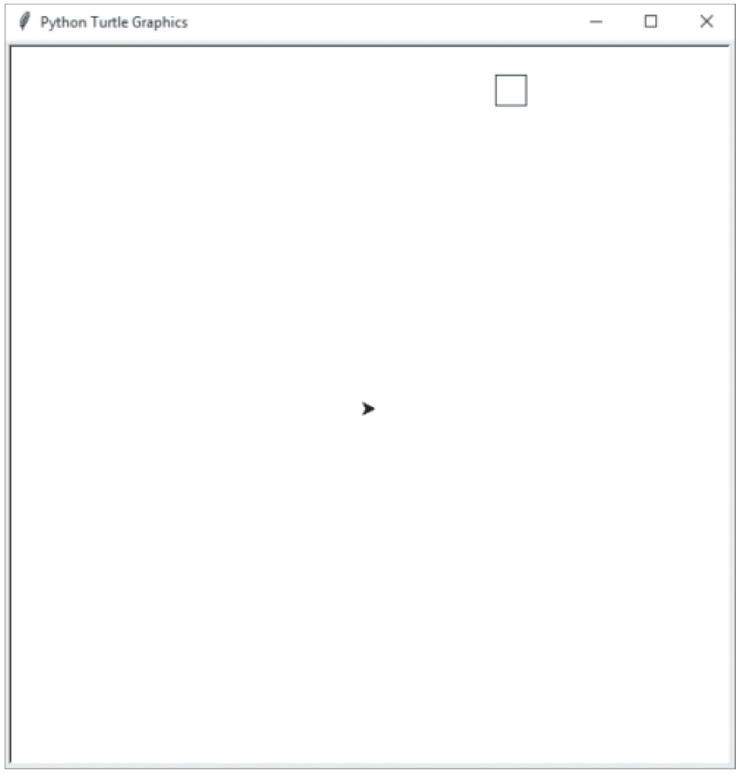
- Determining the Pen Size

```
if turtle.pensize() < 3:
    turtle.pensize(3)
```

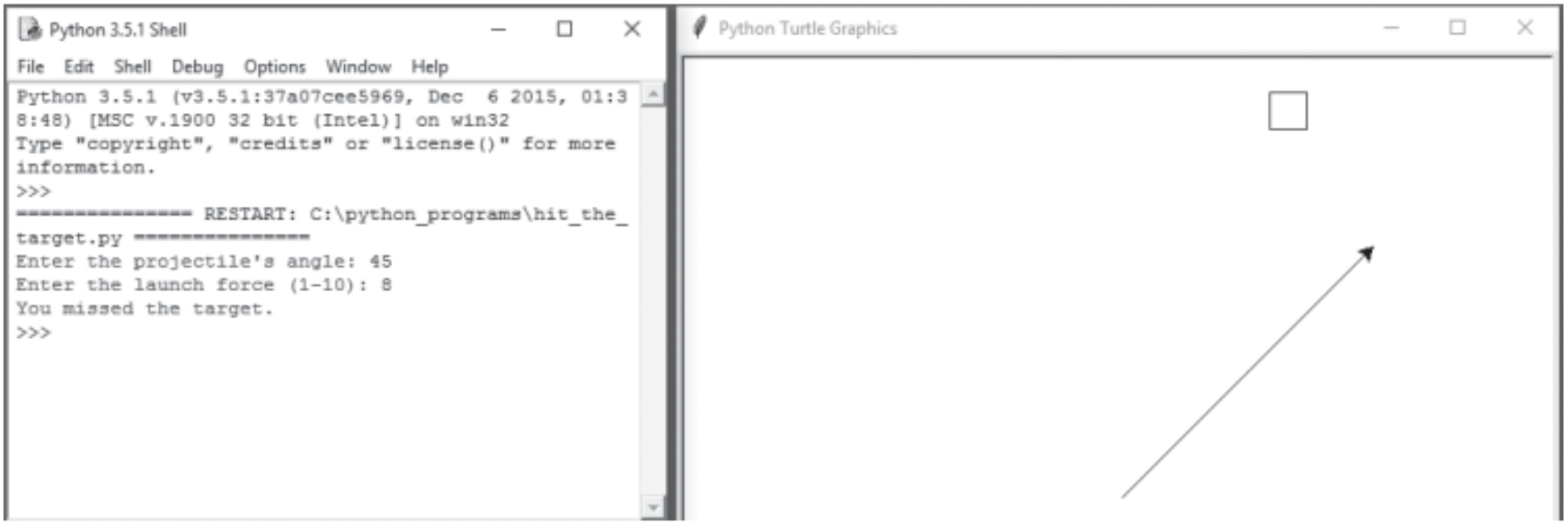
- Determining the Turtle's Animation Speed

```
if turtle.speed() == 0:
    turtle.pencolor('red')
elif turtle.speed() > 5:
    turtle.pencolor('blue')
else:
    turtle.pencolor('green')
```

Hit the target game



- The program in which we entered 45 as the angle, and 8 as the force value. As you can see, the projectile (the turtle) missed the target.



- we ran the program again, entering 67 as the angle and 9.8 as the force value. These values caused the projectile to hit the target.