



CHAPTER 4 SELECTIONS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)

MOTIVATION

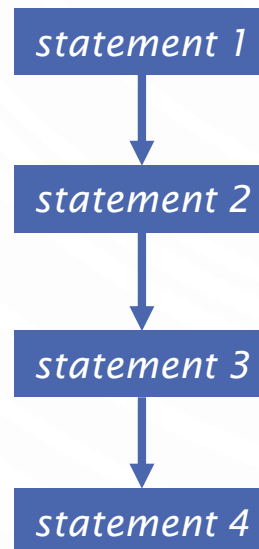
- If you assigned a negative value for radius in πr^2 , the program would print an invalid result. If the radius is negative, you don't want the program to compute the area. How can you deal with this situation?
- Say you run a casino, how would you determine if a slot machine yielded a win or a loss?
- You work for the government, how might you decide if a person is eligible for a tax refund or not?
- Moreover, how would you make a program do something over-and-over, like maintain a set of accounts for a bank?

CONTROL FLOW

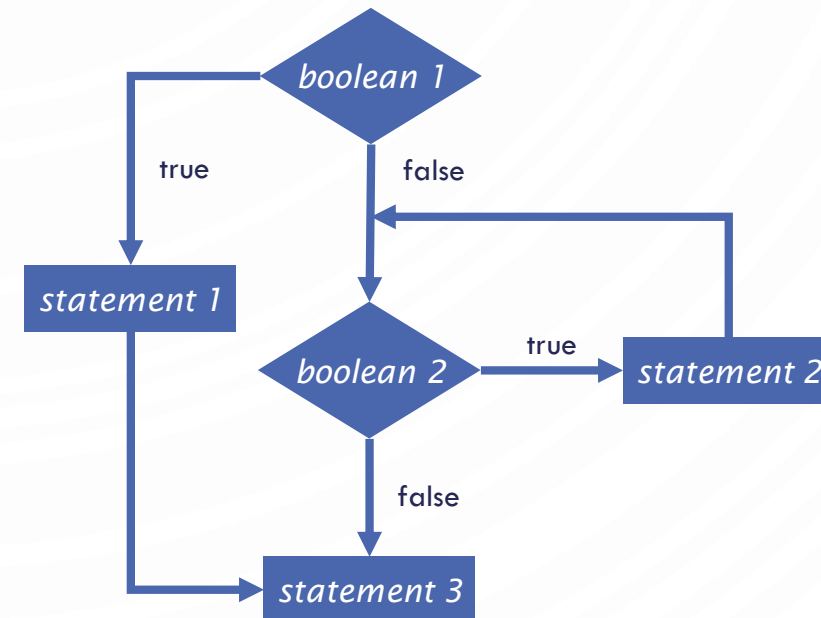
- Control flow.
 - Sequence of statements that are actually executed in a program.
 - Conditionals and loops: enable us to choreograph control flow.

Notation

- Block – statement of code
- Diamond – conditional
- Open circle – start/end of algorithm



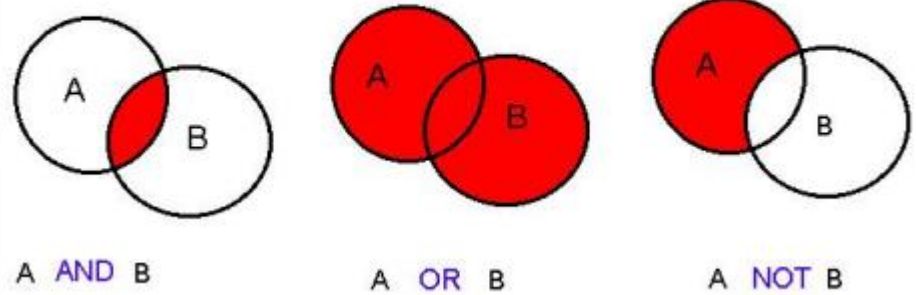
straight-line control flow



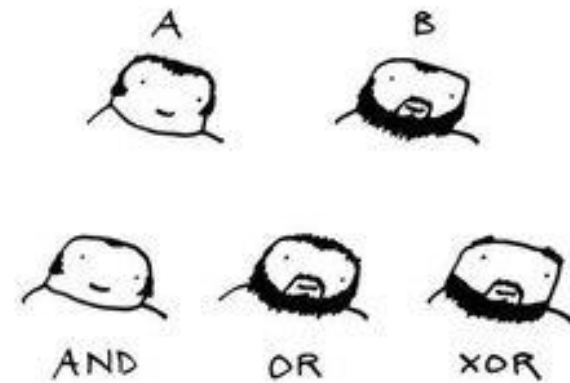
control flow with conditionals and loops

BOOLEANS

BOOLEAN OPERATORS



BOOLEAN HAIR LOGIC



THE BOOLEAN TYPE AND OPERATORS

- Often in a program you need to compare two values, such as whether i is greater than j . There are six comparison operators (also known as **relational operators**) that can be used to compare two values. The result of the comparison is a **Boolean value**: true or false.
- `b = (1 > 2) ;`

RELATIONAL OPERATORS

Python Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
<	<	less than	<code>radius < 0</code>	<code>false</code>
<=	=	less than or equal to	<code>radius <= 0</code>	<code>false</code>
>	>	greater than	<code>radius > 0</code>	<code>true</code>
>=	=	greater than or equal to	<code>radius >= 0</code>	<code>true</code>
==	=	equal to	<code>radius == 0</code>	<code>false</code>
!=	?	not equal to	<code>radius != 0</code>	<code>true</code>

LOGICAL OPERATORS

Python Operator	Math Symbol	Description
<code>not</code>	\neg	logical negation
<code>and</code>	\wedge	logical conjunction
<code>or</code>	\vee	logical disjunction

TRUTH TABLE FOR OPERATOR NOT

p	$\text{not } p$	Example (assume age = 24, weight = 140)
True	False	$\neg(\text{age} > 18)$ is false, because $(\text{age} > 18)$ is true.
False	True	$\neg(\text{weight} = 150)$ is true, because $(\text{weight} = 150)$ is false.

TRUTH TABLE FOR OPERATOR AND

p_1	p_2	p_1 and p_2	Example (assume age = 24, weight = 140)
False	False	False	$(age \leq 18) \wedge (weight < 140)$ is false
False	True	False	$(age \leq 18) \wedge (weight \leq 140)$ is false
True	False	False	$(age > 18) \wedge (weight < 140)$ is false
True	True	True	$(age > 18) \wedge (weight \leq 140)$ is true

TRUTH TABLE FOR OPERATOR OR

p_1	p_2	$p_1 \text{ or } p_2$	Example (assume age = 24, weight = 140)
False	False	False	$(age \leq 18) \vee (weight < 140)$ is false
False	True	True	$(age \leq 18) \vee (weight \leq 140)$ is true
True	False	True	$(age > 18) \vee (weight < 140)$ is true
True	True	True	$(age > 18) \vee (weight \leq 140)$ is true

EXERCISE

- Draw the truth table for $(p_1 \text{ and not } p_2) \text{ or } (\text{not } p_1 \text{ and } p_2)$
- This operation is called exclusive or (XOR)

p_1	p_2	$(p_1 \text{ and not } p_2) \text{ or } (\text{not } p_1 \text{ and } p_2)$
False	False	?
False	True	?
True	False	?
True	True	?

EXAMPLE PROGRAM

- Here is a program that check division by 2 and 3, 2 or 3, and 2 or 3 exclusive

```
1. x = eval(input("Enter a number: "))
2.
3. divBy2 = x % 2 == 0
4. divBy3 = x % 3 == 0
5.
6. print(x, "divisible by 2 and 3: ", divBy2 and divBy3)
7. print(x, "divisible by 2 or 3: ", divBy2 or divBy3)
8. print(x, "divisible by 2 xor 3: ",
      (not divBy2 and divBy3) or (divBy2 and not divBy3))
```

EXERCISE

- Let a user enter a year, and output whether or not it is a leap year. A year is a leap year if it is
 - Divisible by 4 but not by 100
 - OR
 - Divisible by 400
- Do not use any `if` statements, only Boolean expressions

OPERATOR PRECEDENCE

1. +, - (unary +/-)
2. ** (exponentiation)
3. not
4. *, /, //, % (multiplication/division)
5. +, - (addition/subtraction)
6. <, <=, >, >= (comparison)
7. ==, != (equality)
8. and
9. or
10. =, +=, -=, *=, /=, //=, %= (Assignment operator)

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.)
When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.
- If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

EXAMPLE

- Applying the operator precedence and associativity rule, the expression

$$3 + 4 * 4 > 5 * (4 + 3) - 1$$

is evaluated as follows:

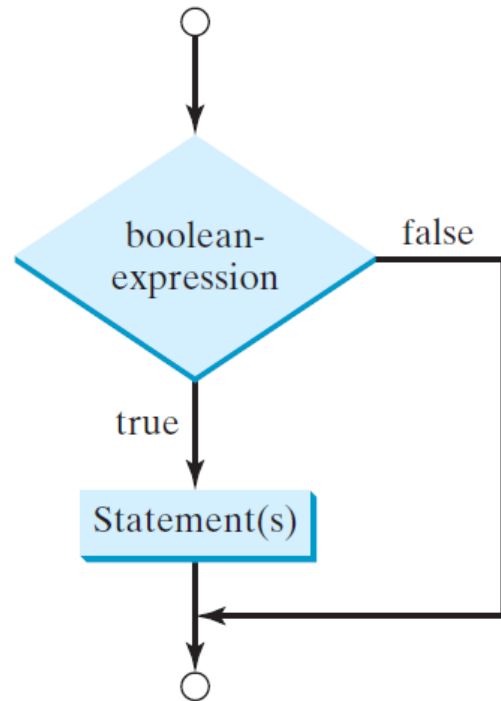
3 + 4 * 4 > 5 * (4 + 3) - 1
3 + 4 * 4 > 5 * 7 - 1 (1) inside parentheses first
3 + 16 > 5 * 7 - 1 (2) multiplication
3 + 16 > 35 - 1 (3) multiplication
19 > 35 - 1 (4) addition
19 > 34 (5) subtraction
false (6) greater than

SELECTIONS AKA CONDITIONALS

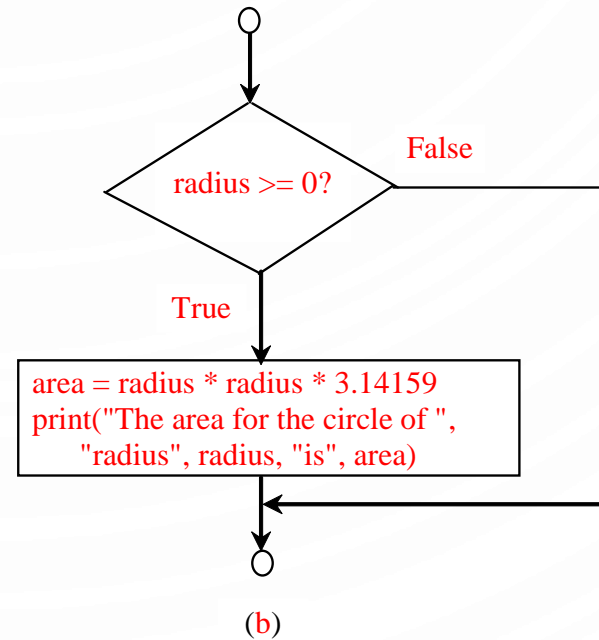


ONE-WAY IF STATEMENTS

1. `if boolean-expression:`
2. `statement(s)`



1. `if radius >= 0:`
2. `area = radius * radius * 3.14159`
3. `print("The area for the circle of radius", radius, "is", area)`



NOTE

- Indentation matters

```
if i > 0:  
print("i is positive")
```

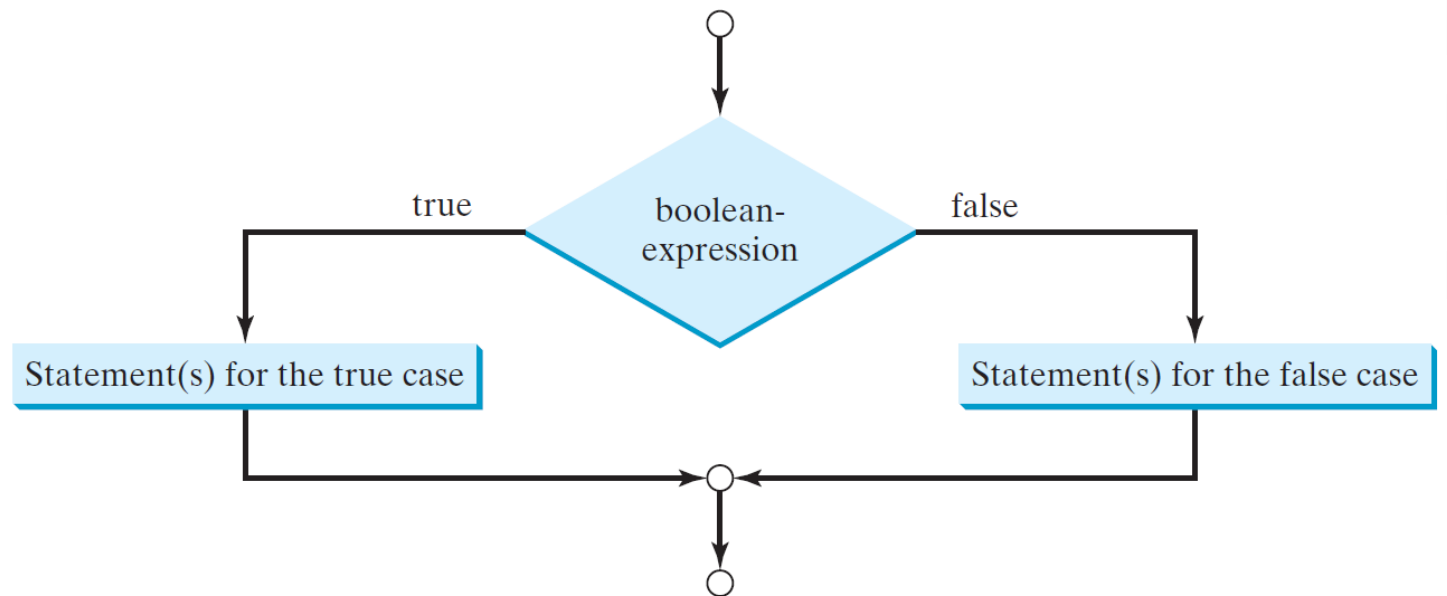
(a) Wrong

```
if i > 0:  
    print("i is positive")
```

(b) Correct

THE TWO-WAY IF STATEMENT

1. `if` *boolean-expression*:
2. *statement(s) -for-the-true-case*
3. `else`:
4. *statement(s) -for-the-false-case*



IF-ELSE EXAMPLE

```
1. if radius >= 0:  
2.     area = radius * radius * 3.14159  
3.     print("The area for the circle of radius",  
4.         radius, " is ", area)  
5. else:  
6.     print("Negative input")
```

EXERCISE: GUESSING GAME

- Use `random.randint(1, 99)` to generate a random number between 1 and 99
 - Remember to `import random`
- Have a user guess the number. If the user is correct output a winning message, otherwise output a losing message
- Lets solve together. Program along with me.

MULTIPLE ALTERNATIVE IF STATEMENTS

Note the syntax for else-if statements

```
if score >= 90.0:  
    grade = 'A'  
else:  
    if score >= 80.0:  
        grade = 'B'  
    else:  
        if score >= 70.0:  
            grade = 'C'  
        else:  
            if score >= 60.0:  
                grade = 'D'  
            else:  
                grade = 'F'
```

(a)

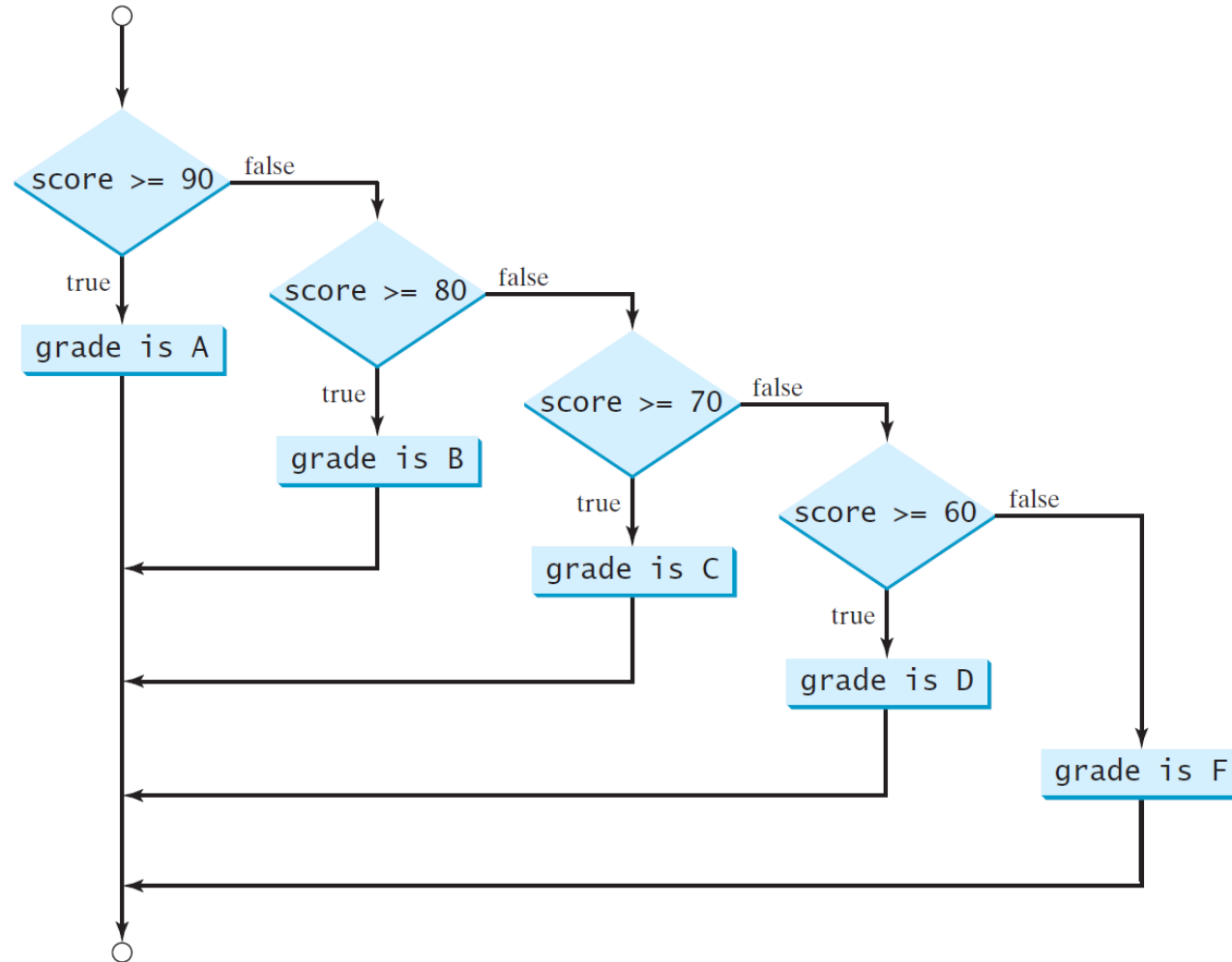
Equivalent

This is better

```
if score >= 90.0:  
    grade = 'A'  
elif score >= 80.0:  
    grade = 'B'  
elif score >= 70.0:  
    grade = 'C'  
elif score >= 60.0:  
    grade = 'D'  
else:  
    grade = 'F'
```

(b)

MULTI-WAY IF-ELSE STATEMENTS



TRACE IF-ELSE STATEMENT

Suppose score is 72.3

```
1.  if score >= 90.0:
2.      grade = 'A'
3.  elif score >= 80.0:
4.      grade = 'B'
5.  elif score >= 70.0:
6.      grade = 'C'
7.  elif score >= 60.0:
8.      grade = 'D'
9.  else:
10.     grade = 'F'
```

Condition is false

TRACE IF-ELSE STATEMENT

Suppose score is 72.3

```
1.  if score >= 90.0:
2.      grade = 'A'
3.  elif score >= 80.0:
4.      grade = 'B'
5.  elif score >= 70.0:
6.      grade = 'C'
7.  elif score >= 60.0:
8.      grade = 'D'
9.  else:
10.     grade = 'F'
```

Condition is false

TRACE IF-ELSE STATEMENT

Suppose score is 72.3

```
1.  if score >= 90.0:
2.      grade = 'A'
3.      elif score >= 80.0:
4.          grade = 'B'
5.      elif score >= 70.0:
6.          grade = 'C'
7.      elif score >= 60.0:
8.          grade = 'D'
9.      else:
10.         grade = 'F'
```

Condition is true

TRACE IF-ELSE STATEMENT

Suppose score is 72.3

```
1.  if score >= 90.0:
2.      grade = 'A'
3.  elif score >= 80.0:
4.      grade = 'B'
5.  elif score >= 70.0:
6.      grade = 'C'
7.  elif score >= 60.0:
8.      grade = 'D'
9.  else:
10.     grade = 'F'
```

Output "C"

TRACE IF-ELSE STATEMENT

Suppose score is 72.3

```
1.  if score >= 90.0:  
2.     grade = 'A'  
3.  elif score >= 80.0:  
4.     grade = 'B'  
5.  elif score >= 70.0:  
6.     grade = 'C'  
7.  elif score >= 60.0:  
8.     grade = 'D'  
9.  else:  
10.     grade = 'F'
```

Exit the block

PROBLEM: COMPUTING TAXES

- The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for

<i>Marginal Tax Rate</i>	<i>Single</i>	<i>Married Filing Jointly or Qualifying Widow(er)</i>	<i>Married Filing Separately</i>	<i>Head of Household</i>
10%	\$0 – \$8,350	\$0 – \$16,700	\$0 – \$8,350	\$0 – \$11,950
15%	\$8,351 – \$33,950	\$16,701 – \$67,900	\$8,351 – \$33,950	\$11,951 – \$45,500
25%	\$33,951 – \$82,250	\$67,901 – \$137,050	\$33,951 – \$68,525	\$45,501 – \$117,450
28%	\$82,251 – \$171,550	\$137,051 – \$208,850	\$68,526 – \$104,425	\$117,451 – \$190,200
33%	\$171,551 – \$372,950	\$208,851 – \$372,950	\$104,426 – \$186,475	\$190,201 – \$372,950
35%	\$372,951+	\$372,951+	\$186,476+	\$372,951+

PROBLEM: COMPUTING TAXES, CONT.

```
1.  if    status == 0:
2.      # Compute tax for single filers
3.  elif status == 1:
4.      # Compute tax for married filing jointly
5.  elif status == 2:
6.      # Compute tax for married filing separately
7.  elif status == 3:
8.      # Compute tax for head of household
9.  else:
10.     # Display wrong status
```

COMMON ERRORS

- Most common errors in selection statements are caused by incorrect indentation. Consider the following code in (a) and (b).

```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
print("The area is", area)
```

(a) Wrong

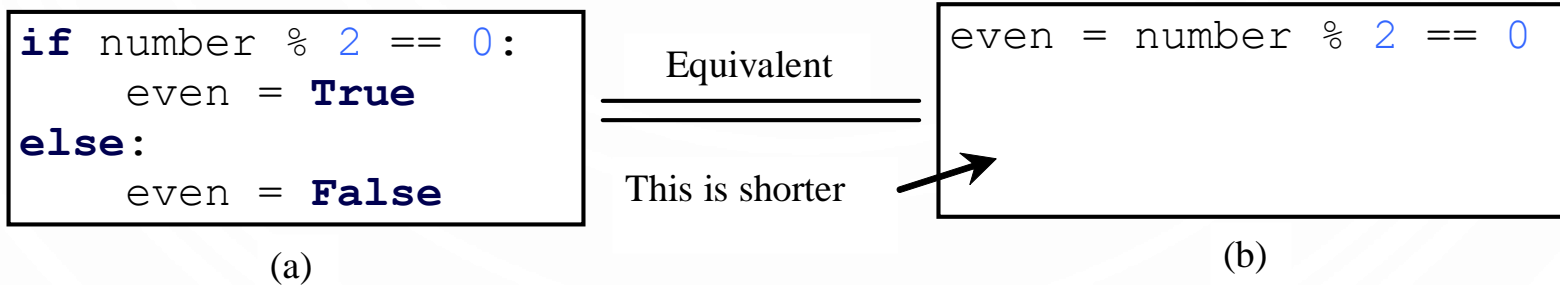
```
radius = -20

if radius >= 0:
    area = radius * radius * 3.14
    print("The area is", area)
```

(b) Correct

TIP

- Use Boolean expressions when you can



Hint 1: If you **LIKE** getting big points off of style. I recommend writing (a)!

Hint 2: Hint 1 is sarcasm...

CAUTION

```
if even == True:  
    print("It is even.");
```

(a)

Equivalent

```
if even:  
    print("It is even.");
```

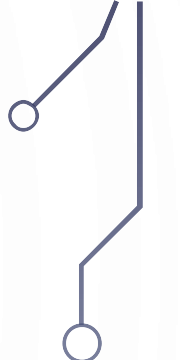
(b)

Hint 1: If you **LIKE** getting big points off of style. I recommend writing (a)!

Hint 2: Hint 1 is sarcasm...



EXERCISE: GUESSING GAME

- Extend the previous guessing game example to allow two guesses, and descriptive messages of a guess being correct, over, or under.
 - Hint: We have 3 conditions per guess...
- 

CONDITIONAL EXPRESSIONS

```
1. if x > 0:  
2.     y = 1  
3. else:  
4.     y = -1
```

- is equivalent to a special **ternary** operator
- $y = 1$ **if** (x > 0) **else** -1
- expression1 **if** (boolean-expression) **else** expression2

EXERCISE: GRAPHICS

- Write a program that prompts the user to enter the center, width, and heights of two rectangles and determines whether the second rectangle is inside the first, overlaps with the first, or does not overlap
 - Display text describing the case in the window output (not the console)
 - Bonus, calculate the rectangular overlap and display it in another color (red)

EXERCISE: DAY OF THE WEEK

- Zeller's congruence is an algorithm to calculate the day of the week:

$$h = \left(d + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + k + \left\lfloor \frac{k}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor + 5c \right) \bmod 7$$

where:

- h is the day of the week (0: Saturday, 1: Sunday, ..., 6: Friday)
 - d is the day of the month
 - m is the month (1: January, 2: February, ..., 12: December)
 - Note: when m is 1 or 2, add 12 and subtract 1 from the year – consider these as months 13/14 of the prior year
 - c is the century (i.e., $\left\lfloor \frac{year}{100} \right\rfloor$)
 - k is the year of the century (i.e., $year \bmod 100$)
- Write a program that prompts for the year, month, and day of the month and then it displays the name of the day of the week. Take the month as a number (1-12).