4.1 If-else branches (general)

Branch concept

People who have worked in restaurants may be familiar with steering people to different-sized tables based on group size.

PARTICIPATION ACTIVITY 4.1.1: Branching concept.

Animation content:

Static figure: The text, Party of:, is shown in front of an image of a person labeled Host. The layout of a restaurant is displayed in rows, where the first row is a rectangle representing the counter, the second row is 3 squares representing small tables, and the third row is 3 rectangles representing large tables.
Step 1: A restaurant host seats patrons. The host seats a party of 1 at the counter. The text, Party of: 1, is displayed, and the number 1 appears on the counter.
Step 2: A party of 2 is seated at a small table. Larger-sized parties are seated at a large table. The text, Party of: 2, is displayed, and the number 2 appears on a small table. The text, Party of: 4, is displayed, and the number 4 appears on a large table. The text, Party of: 1, is displayed, and another number 1 appears on the counter. The text, Party of: 3, is displayed, and the number 3 appears on a large table.
Step 3: The host mentally executes the algorithm: If party of one, seat at the counter; if party of 2, seat at a small table; for parties of 3 or more, seat at a large table. The text, If party of 1:, appears next to the counter. The text, Else If party of 2:, appears next to the row of small tables. The text, Else:, appears next to the row of large tables.

Animation captions:

1. A restaurant host seats patrons. The host seats a party of 1 at the counter.
2. A party of 2 is seated at a small table. Larger-sized parties are seated at a large table.
3. The host mentally executes the algorithm: If party of one, seat at the counter; if party of 2, seat at a small table; for parties of 3 or more, seat at a large table.

PARTICIPATION ACTIVITY 4.1.2: Branch concept.

Consider the example above.

1) A party of 1 is seated at ____ .

- ○ the counter
- ○ a small table

2) A party of 2 is seated at ____ .

- ○ the counter
- ○ a small table

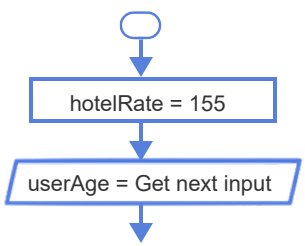
3) A party of 5 is seated ____ .

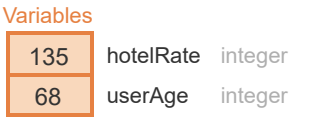
- ○ at a large table
- ○ nowhere

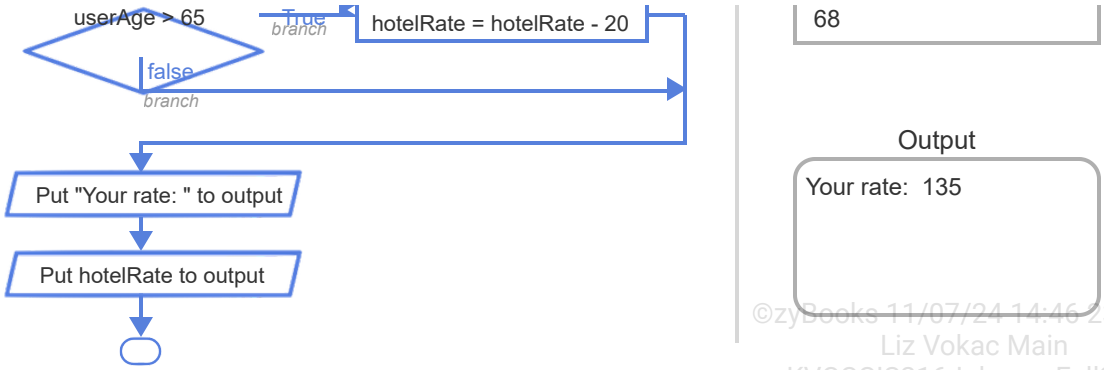
Branch basics (If)

In a program, a ***branch*** is a sequence of statements only executed under a certain condition. Ex: A hotel may discount a price only for people over age 65. An ***if*** branch is a branch taken only *if* an expression is true.

PARTICIPATION ACTIVITY 4.1.3: Branches: Hotel rate example.







Animation content:

The animation executes the following Coral program:

```
integer hotelRate
integer userAge

hotelRate = 155
userAge = Get next input
if userAge > 65
  hotelRate = hotelRate - 20
Put "Your rate: " to output
Put hotelRate to output
```

Variables in memory is as follows:
135 hotelRate: integer
68 userAge: integer

Input is as follows:
68

Output (screen) is as follows:
Your rate: 135

Animation captions:

1. A decision leads to two program branches. If the expression is True, the first branch executes. Otherwise, the second branch executes.
2. If userAge is 68, then 68 > 65 is true. So the first branch executes, which discounts hotelRate.
3. Execution rejoins the other branch and continues with subsequent statements, outputting 135. If userAge were instead 50, the output would be 155.

PARTICIPATION ACTIVITY 4.1.4: Branches.

Consider the hotel rate example above.

1) If userAge is 20, does the True or False branch execute?
   - ○ True branch
   - ○ False branch

2) If userAge is 20, does the executed branch update hotelRate?
   - ○ Yes
   - ○ No

3) If userAge is 20, what hotel rate does the program output?
   - ○ 155
   - ○ 135

4) If userAge is 70, what hotel rate does the program output?
   - ○ 155
   - ○ 135

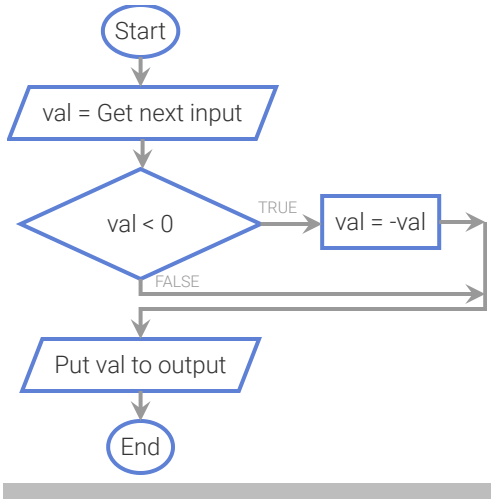
5) Do the last two statements always execute for any value of userAge?
   - ○ Yes
   - ○ No

If branch example: Absolute value

The example below shows how an if branch can be used to compute an absolute value of a number.

PARTICIPATION ACTIVITY 4.1.5: Computing absolute value.

Full screen



Variables

0 val integer

Input

-99

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 4.1.6: Example if branch: Absolute value.

Consider the example above.

1) If the input is -6, does the branch execute?
   - Yes
   - No

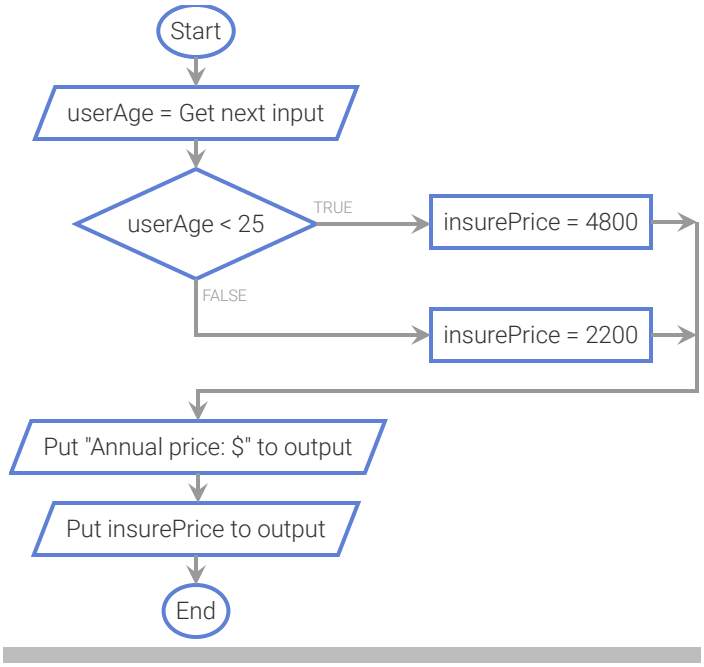
2) If the input is 0, does the branch execute?
   - Yes
   - No

If-else branches

An ***if-else*** branch has two branches: The first branch is executed *if* an expression is true, *else* the other branch is executed.

In the example below, if a user inputs an age less than 25, the statement `insurePrice = 4800` executes. Else, `insurePrice = 2200` executes.

PARTICIPATION ACTIVITY 4.1.7: Insurance price. Full screen



Variables

0	userAge	integer
0	insurePrice	integer

Input

22

Output

ENTER EXECUTION STEP RUN Execution speed Medium

Car insurance prices

(Car insurance prices for drivers under 25 are higher because 1 in 6 such drivers are involved in an accident each year vs. 1 in 15 for older drivers. Source: www.census.gov, 2009).

PARTICIPATION ACTIVITY 4.1.8: If-else branches.

Consider the insurance price example above.

1) If userAge is 18, what price is output?

- 4800
- 2200

2) If userAge is 30, what price is output?

- 4800
- 2200

3) If userAge is 25, what price is output?

- 4800
- 2200

4) For what value of userAge will both branches execute?

- 15
- 25
- None

5) For what value of userAge will neither branch execute?

- 30
- 25
- None

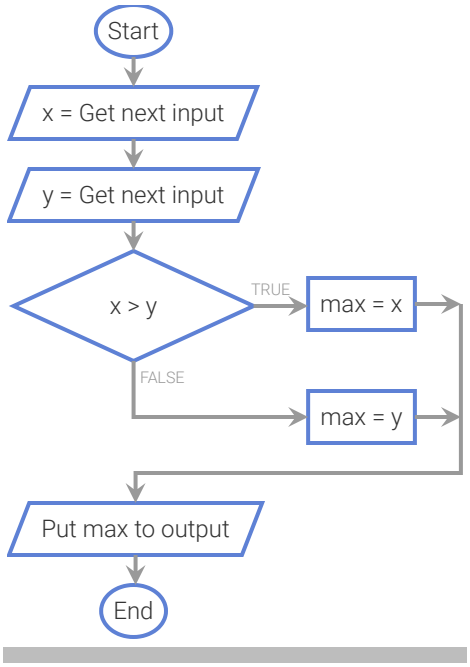
6) For what value of userAge will the output statements not execute?

- 20
- 25
- None

If-else example: Max

The example below shows how an if-else can be used to get the maximum of two values.

PARTICIPATION ACTIVITY 4.1.9: If-else branches example: Max. Full screen



Variables

0	x	integer
0	y	integer
0	max	integer

Input

55 79

Output

ENTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 4.1.10: If-else example: Max.

Consider the example above.

1) When the input is -3 0, which branch executes?
   - If
   - Else

2) When the input is 99 98, which branch executes?
   - If
   - Else

3) The if branch assigns max = x. The else branch assigns max = ?
   - x
   - y

4) If the inputs are 5 5, does max get assigned with x or y?
   - x

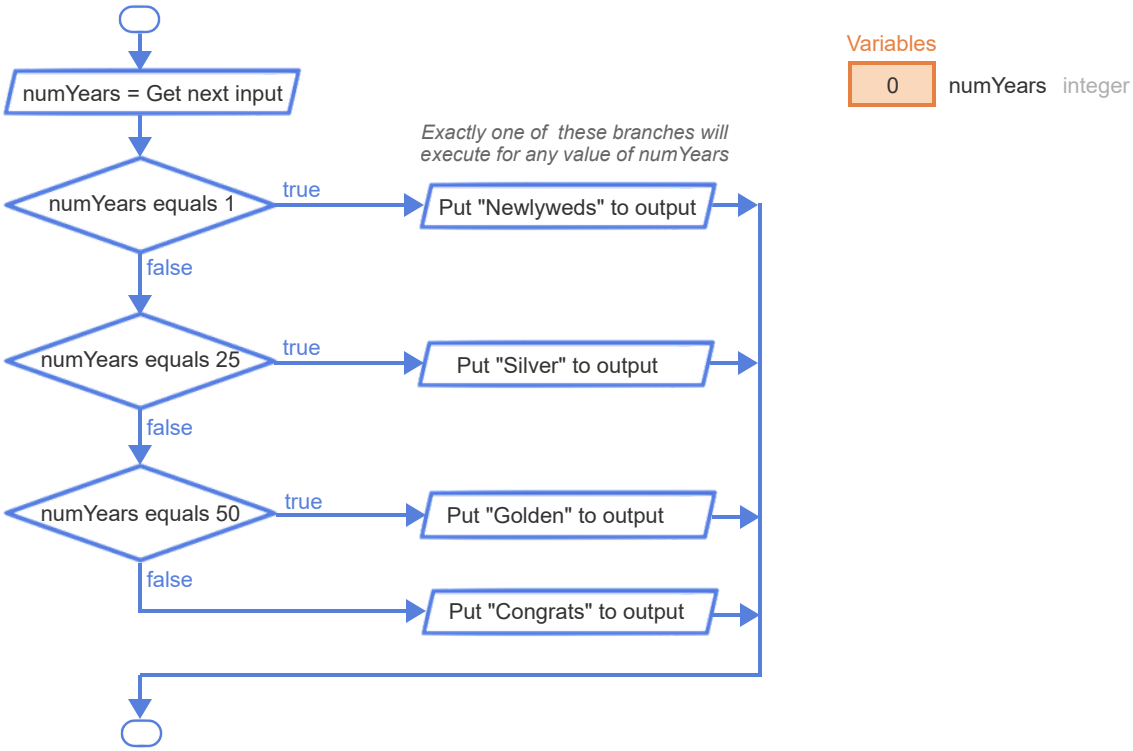
y

If-elseif-else branches

Commonly a programmer wishes to take one of multiple (three or more) branches. An if-else can be extended to an if-elseif-else structure. Each branch's expression is checked in sequence; as soon as one branch's expression is found to be true, that branch is taken. If no expression is found true, execution will reach the else branch, which then executes.

Note: The else part is optional. If omitted, then if none of the previous expressions are true, no branch executes.

PARTICIPATION ACTIVITY 4.1.11: If-elseif example: Anniversaries.



Animation content:

The animation executes the following Coral program:

integer numYears

numYears = Get next input

```
if numYears == 1
  Put "Newlyweds" to output
else
  if numYears == 25
    Put "Silver" to output
  else
    if numYears == 50
      Put "Golden" to output
    else
      Put "Congrats" to output
```


Variables in memory is as follows:
0 numYears: integer

Animation captions:

1. This program detects the specific value of a variable. If numYears is 1, the first branch executes and "Newlyweds" is output.
2. If numYears is 25, the second branch executes and "Silver" is output. Otherwise, if numYears is 50, the third branch executes and "Golden" is output.
3. Otherwise, the last branch executes.

PARTICIPATION ACTIVITY 4.1.12: If-elseif-else.

Consider the if-elseif-else structure below:

```
if x equals -1
     Put "Disagrees" to output
else if x equals 0
     Put "Neutral" to output
else if x equals 1
     Put "Agrees" to output
else
     Put "Invalid entry" to output
```

1) If x is 1, what is the output?

- Disagrees
- Neutral
- Agrees
- Invalid entry

2) If x is -2, what is output?

- Disagrees

- Invalid entry
- (Nothing is output)

3) Could the programmer have written the three branches in the order x equals 1, x equals 0, and x equals -1, and achieved the same results?

- No
- Yes

4) In the code above, suppose a programmer, after the third branch (x equals 1), inserts a new branch: else if x equals -1. When might that new branch execute?

- When x is -1
- When x is 1
- Never

5) In the code above, suppose a programmer removed the else part entirely. If x is 2, which is correct?

- The last branch, meaning the else if x equals 1 branch, will execute.
- No branch will execute.
- The program is not legal.

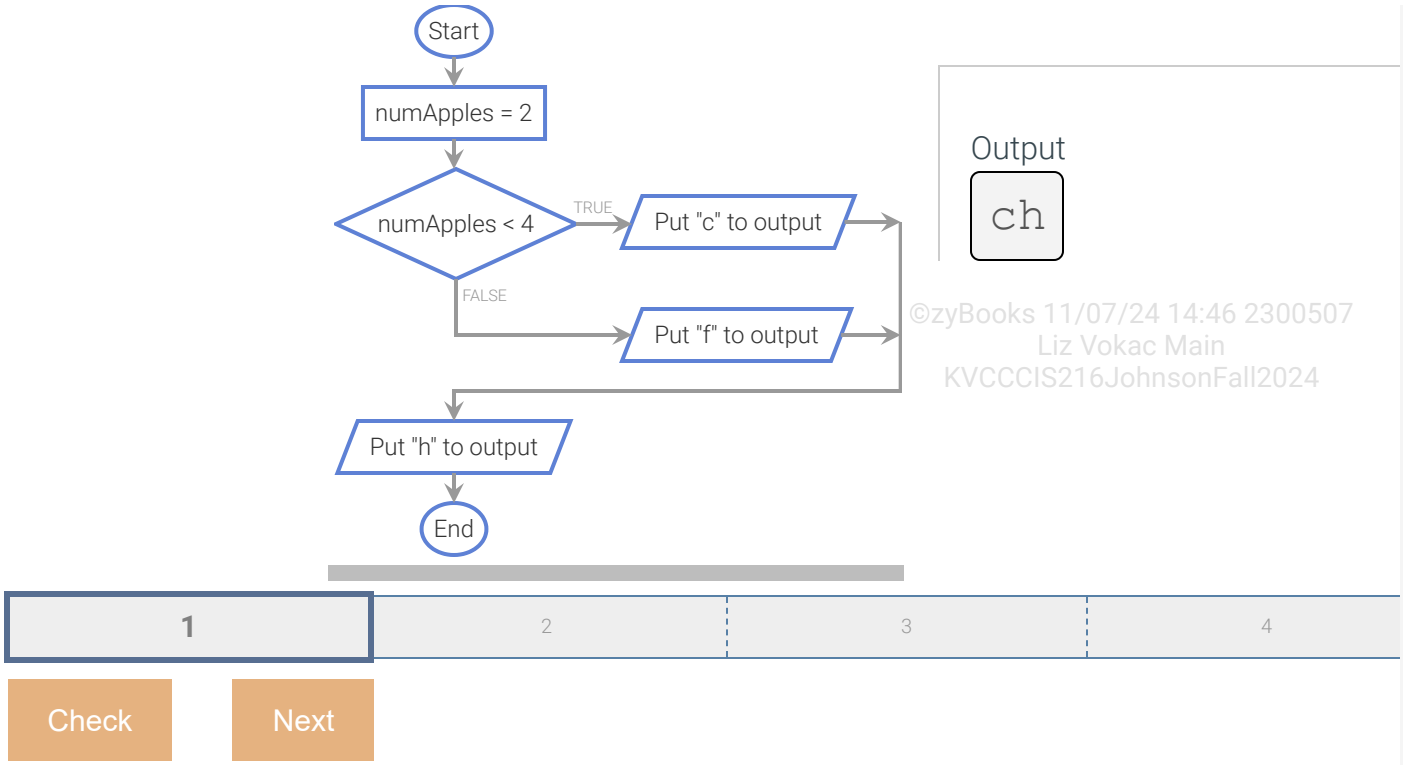
CHALLENGE ACTIVITY 4.1.1: If-else branches.

Note: Level 5 uses ==, known as the equality operator, to mean equals. Ex: `x == 9` means `x equals 9`.

566436.4601014.qx3zqy7

Start

Type the program's output



4.2 Detecting equal values with branches

Detecting if two items are equal using an if statement

A program commonly needs to determine if two items are equal. Ex: If a hotel gives a discount for guests on their 50th wedding anniversary, a program to calculate the discount can check if a variable num_years is equal to the value 50. A programmer can use an if statement to check if two values are equal.

An ***if*** statement executes a group of statements if an expression is true. The statements in a branch must be indented, typically four spaces.

The example below uses ==. The ***equality operator*** (***==***) evaluates to True if the left and right sides are equal. Ex: If num_years is 50, then num_years == 50 evaluates to true. Note the equality operator is ==, not =.

PARTICIPATION ACTIVITY 4.2.1: Detecting if two items are equal: Hotel discount.

```
hotel_rate = 150

num_years = int(input('Enter years married: '))
```

Memory

95

```
if num_years == 50:
   print('Congratulations on 50 years of marriage!')
   hotel_rate = hotel_rate / 2

print(f'Your hotel rate: ${hotel_rate:.2f}')
```

96	75.0	hotel_rate
97	50	num_years
98		

```
hotel_rate = 150

num_years = int(input('Enter years married: '))

if num_years == 50:
   print('Congratulations on 50 years of marriage!')
   hotel_rate = hotel_rate / 2

print(f'Your hotel rate: ${hotel_rate:.2f}')
```

```
Enter years married: 50
Congratulations on 50 years of marriage
Your hotel rate: $75.00
```


Animation content:

The program shown:
hotel_rate = 150

num_years = int(input('Enter years married: '))

if num_years == 50:
   print('Congratulations on 50 years of marriage!')
   hotel_rate = hotel_rate / 2

print(f'Your hotel rate: ${hotel_rate:.2f}')

The console input/output shown:
Enter years married: 50
Congratulations on 50 years of marriage!
Your hotel rate: $75.00

Animation captions:

1. An if statement executes a group of statements if an expression is True. The program assigns hotel_rate with 150 and then retrieves the number of years the user has been married from input.
2. num_years is 50. So the expression num_years == 50 evaluates to True, and the if statement will execute. The statements after the colon : will execute next.
3. hotel_rate is divided in half, which is the discount for guests celebrating their 50th wedding anniversary.
4. The program completes by printing the hotel rate.

PARTICIPATION ACTIVITY 4.2.2: If statement.

What is the final value of num_items?

1)
```
bonus_val = 10
num_items = 1

if bonus_val == 10:
    num_items = num_items +
3
```


Check Show answer

2)
```
bonus_val = 0
num_items = 1

if bonus_val == 10:
    num_items = num_items +
3
```

Check Show answer

Equality and inequality operators

Whereas the equality operator checks whether two values are equal, the ***inequality operator*** (***!=***) evaluates to True if the left and right sides are not equal, or different.

An expression involving an equality or inequality operator evaluates to a Boolean value. A ***Boolean*** is a type that has just two values: True or False.

Table 4.2.1: Equality and inequality operators.

Equality operators	Description	Example (assume x is 3)
==	a ***==*** b means a is equal to b.	x == 3 is True x == 4 is False

!=	a **!=** b means a is not equal to b	x != 3 is False x != 4 is True

PARTICIPATION ACTIVITY 4.2.3: Evaluating expressions that have equality operators.

Indicate whether the expression evaluates to True or False.
x is 5, y is 7.

1) x == 5

- True
- False

2) x == y

- True
- False

3) y != 7

- True
- False

4) y != 99

- True
- False

5) x != y

- True
- False

PARTICIPATION ACTIVITY 4.2.4: Creating expressions with equality operators.

Type the equality or inequality operator to make the expression true.

1) num_dogs is 0.

```
num_dogs  [ ]  0
```

Check Show answer

2) num_dogs and num_cats are the same.

`num_dogs` [ ] `num_cats`

Check Show answer

3) num_dogs and num_cats differ

`num_dogs` [ ] `num_cats`

Check Show answer

4) num_dogs is either less than or greater than num_cats.

`num_dogs` [ ] `num_cats`

Check Show answer

5) user_char is the character 'x'.

`user_char` [ ] `'x'`

Check Show answer

If-else statement

An ***if-else*** statement executes one group of statements when an expression is true, and another group of statements when the expression is false. In the example below, the if-else statement outputs if a number entered by the user is even or odd. The if statement executes if divRemainder is equal to 0, and the else statement executes if divRemainder is not equal to 0.

PARTICIPATION ACTIVITY 4.2.5: If-else statement: Determining if a number is even or odd.


```
user_num = int(input('Enter a number: '))

div_remainder = user_num % 2

if div_remainder == 0:
```

Memory

95		
96	1	div_remainder
97	45	user_num

```
    print(f'{user_num} is even.')
else:
    print(f'{user_num} is odd.')
```

98

Enter a number: 22
22 is even.

Enter a number: 45
45 is odd.

Animation content:

The program shown:
user_num = int(input('Enter a number: '))

div_remainder = user_num % 2

if div_remainder == 0:
  print(user_num, 'is even.')
else:
  print(user_num, 'is odd.')

Console input/output shown from 2 runs of the program:
Enter a number: 22
22 is even.

Enter a number: 45
45 is odd.

Animation captions:

1. An if-else statement executes a group of statements if an expression is True, and executes another group of statements otherwise.
2. user_num % 2 evaluates to the remainder of dividing user_num by 2. user_num is 22, so div_remainder is assigned with 0.
3. The if statement's expression div_remainder == 0 evaluates to 0 == 0, which is True. So the if statements execute.
4. user_num is 45, so div_remainder is assigned with 1. The if statement's expression div_remainder == 0 evaluates to 1 == 0, which is False. So the else's statements execute.

PARTICIPATION ACTIVITY 4.2.6: If-else statements.

1) What is the final value of num_items?

```
bonus_val = 12

if bonus_val == 12:
    num_items = 100
else:
    num_items = 200
```

Check Show answer

2) What is the final value of num_items?

```
bonus_val = 11

if bonus_val == 12:
    num_items = 100
else:
    num_items = 200
```

Check Show answer

3) What is the final value of num_items?

```
bonus_val = 15
num_items = 44

if bonus_val == 14:
    num_items = num_items +
3
else:
    num_items = num_items +
6

num_items = num_items + 1
```

Check Show answer

4) What is the final value of bonus_val?

```
bonus_val = 11

if bonus_val != 12:
   bonus_val = bonus_val + 1
else:
   bonus_val = bonus_val +
10
```

Check Show answer

5) What is the final value of bonus_val?

```
bonus_val = 12

if bonus_val == 12:
    bonus_val = bonus_val +
2
    bonus_val = 3 *
bonus_val

else:
    bonus_val = bonus_val +
10
```

Check Show answer

PARTICIPATION ACTIVITY 4.2.7: Writing an if-else statement.

Translate each description to an if-else statement as directly as possible. (Not checked, but please indent a branch's statements a consistent number of spaces, such as three spaces.)

1) If user_age equals 62, assign item_discount with 15. Else, assign item_discount with 0.

Check Show answer

2) If num_people equals 10, execute group_size = 2 * group_size. Otherwise, execute group_size = 3 * group_size and num_people = num_people - 1.

Check Show answer

3) If num_players does not equal 11, execute team_size = 11. Otherwise, execute team_size = num_players. Then, no matter the value of num_players, execute team_size = 2 * team_size.

Check Show answer

CHALLENGE ACTIVITY 4.2.1: Branches with equality and inequality operators: Enter the output.

566436.4601014.qx3zqy7

Start

Type the program's output

```
num_puppies = 5

if num_puppies == 5:
   print('c')
else:
   print('d')

print('h')
```

c
h

1

Check Next

CHALLENGE ACTIVITY 4.2.2: Basic if-else.

566436.4601014.qx3zqy7

Start

Write an if-else statement for the following:

If barcode_check_digit is equal to 7, execute group_id = 1. Else, execute group_id = barcode_check_digit.

Ex: If barcode_check_digit is 3, then group_id = 3.

```
1 barcode_check_digit = int(input())  # Program will be tested with values: 5, 6,
2
3 ''' Your code goes here '''
4
5 print(group_id)
6
```

1

Check Next

Multi-branch if-else statements

Commonly, a program may need to detect several specific values of a variable. An if-else statement can be extended to have three (or more) branches. Additional branches use the ***elif*** keyword, which means "else if". Each branch's expression is checked in sequence. As soon as one branch's expression is found to be True, that branch's statement executes (and no subsequent branch is considered). If no expression is True, the else branch executes. The example below detects values of 1, 25, or 50 for variable num_years.

Figure 4.2.1: Multi-branch if-else statement. Only 1 branch will execute.

```
if expression1:
   # Statements that execute when expression1 is True
   # (first branch)
elif expression2:
   # Statements that execute when expression1 is False and expression2 is
True
   # (second branch)
else:
   # Statements that execute when expression1 is False and expression2 is
False
   # (third branch)
```

Figure 4.2.2: Multi-branch if-else example: Anniversaries.

```
num_years = int(input('Enter number years
married: '))

if num_years == 1:
    print('Your first year -- great!')
elif num_years == 10:
    print('A whole decade -- impressive.')
elif num_years == 25:
    print('Your silver anniversary -- enjoy.')
elif num_years == 50:
    print('Your golden anniversary -- amazing.')
else:
    print('Nothing special.')
```

```
Enter number years married:
10
A whole decade -- impressive.

...

Enter number years married:
25
Your silver anniversary --
enjoy.

...

Enter number years married:
30
Nothing special.

...

Enter number years married: 1
Your first year -- great!
```

PARTICIPATION ACTIVITY 4.2.8: Multi-branch if-else statements.

What is the final value of employee_bonus for each given value of num_sales?

```
if num_sales == 0:
    employee_bonus = 0
elif num_sales == 1:
    employee_bonus = 2
elif num_sales == 2:
    employee_bonus = 5
else:
    employee_bonus = 10
```


1) num_sales is 2

Check Show answer

2) num_sales is 0

Check Show answer

3) num_sales is 7

Check Show answer

CHALLENGE ACTIVITY 4.2.3: Detect specific values.

566436.4601014.qx3zqy7

Start

If http_response_code is 409, output 'Conflict'. Otherwise, output 'No conflict'.

▶ **Click here for examples**

```
1 http_response_code = int(input())
2
3 ''' Your code goes here '''
4
```

1 2

Check Next level

CHALLENGE ACTIVITY 4.2.4: Basic if-else expression.

Write an expression so that "User is 18" is printed if the value of user_age is equal to 18. Write *only* the expression.

Ex: If the input is `18`, then the output is:

```
User is 18
```

Ex: If the input is `30`, then the output is:

```
Not 18
```

Note: Replace `''' Your solution goes here '''` (including quotes) with your solution.

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 user_age = int(input())
2 if ''' Your solution goes here ''':
3     print('User is 18')
4 else:
5     print('Not 18')
```

Run

View your last submission

4.3 Detecting ranges with branches (general)

Detecting ranges using if-elseif-else

A common programming task is to detect if a value lies within a certain range and then perform an action depending on where the value lies. Ex: If Timmy is less than 6, he can play pee-wee soccer. If Timmy is between 6 and 17, he can play junior league soccer, and if he's older than 17, he can play professional soccer.

An if-elseif-else structure can detect number ranges with each branch performing a different action for each range. Each expression only needs to indicate the upper range part; if execution reaches an expression, the lower range part is implicit from the previous expressions being False.

PARTICIPATION ACTIVITY 4.3.1: An if-elseif-else structure can elegantly detect ranges.

```
1
2
3
4
5   5 or under    No teams     If age < 6:      No teams
6
7   Under 8       6, 7         Else If age < 8: Play on U8 team
8
9   Under 10      8, 9         Else If age < 10: Play on U10 team
10
11  Under 12      10, 11       Else If age < 12: Play on U12 team
12  12 or over    No teams     Else:            No teams
13
```

Animation content:

Static figure: A column containing the numbers 1 through 13, representing kids ages, is displayed.
Step 1: Kids of various ages may wish to play soccer. A soccer club may not have teams for kids 5 and under. The numbers 1 through 5 are highlighted, and the text, 5 or under: No teams, appears next to the number 5.
Step 2: One level of teams is listed as "Under 8" (or just U8), which is understood to mean just 7 or 6, but not 5 or younger. The numbers 6 and 7 are highlighted, and the text, Under 8: 6, 7, appears next to the number 7.
Step 3: Likewise, U10 means 9 and 8, and U12 means 11 and 10. No teams exist for ages 12 and over. The numbers 8 and 9 are highlighted, and the text, Under 10: 8, 9, appears next to the number 9. The numbers 10 and 11 are highlighted, and the text, Under 12: 10, 11, appears next to the number 11. The numbers 12 and 13 are highlighted, and the text, 12 or over: No teams, appears next to the number 12.
Step 4: An if-elseif-else structure can elegantly capture such ranges. When an expression is checked, the reviewer knows that all previous expressions are False, thus defining the low-range end. The text, If age < 6: No teams, appears next to the number 5. The text, Else If age < 8: Play on U8 team, appears next to the number 7. The text, Else If age < 10: Play on U10 team, appears next to the number 9. The text, Else If age < 12: Play on U12 team, appears next to the number 11. The text, Else: No teams, appears next to the number 12.

Animation captions:

1. Kids of various ages may wish to play soccer. A soccer club may not have teams for kids 5 and under.
2. One level of teams is listed as "Under 8" (or just U8), which is understood to mean just 7 or 6, but not 5 or younger.
3. Likewise, U10 means 9 and 8, and U12 means 11 and 10. No teams exist for ages 12 and over.
4. An if-elseif-else structure can elegantly capture such ranges. When an expression is checked, the reviewer knows that all previous expressions are False, thus defining the low-range end.

PARTICIPATION ACTIVITY 4.3.2: Using if-elseif-else to detect increasing ranges.

Indicate the range corresponding to each branch. x is a nonnegative integer.

If unable to drag and drop, refresh the page.

0-9 | 20-29 | 10-19 | 30+

	If x < 10 : Branch 1

	Else if x < 20 : Branch 2
	Else if x < 30 : Branch 3
	Else : Branch 4

Reset

PARTICIPATION ACTIVITY 4.3.3: More ranges with if-elseif-else.

Indicate the range detected by the expression, assuming each question continues a single if-elseif-else structure. Type ranges as: 25-29

1) If x > 100 : Branch 1

`- infinity`

Check Show answer

2) Else if x > 50 : Branch 2

Check Show answer

3) Else

`-infinity -`

Check Show answer

4) Is this a reasonable if-elseif-else structure? Type yes or no.

If x < 100: Branch 1
Else If x < 200: Branch 2
Else If x < 150: Branch 3
Else: Branch 4

Check Show answer

CORAL (pseudocode) comments

This section uses a pseudocode language called CORAL. Unlike Python comments that begin with #, CORAL comments begin with //.

CHALLENGE ACTIVITY 4.3.1: Decision sequence to detect increasing ranges.

566436.4601014.qx3zqy7

Start

Indicate the range covered by the following decision. Assume x is a nonnegative integer.

```
x < 13
    // Range covered: Ex: 5 - Ex: 5
x < 28

Else
```

1 2 3

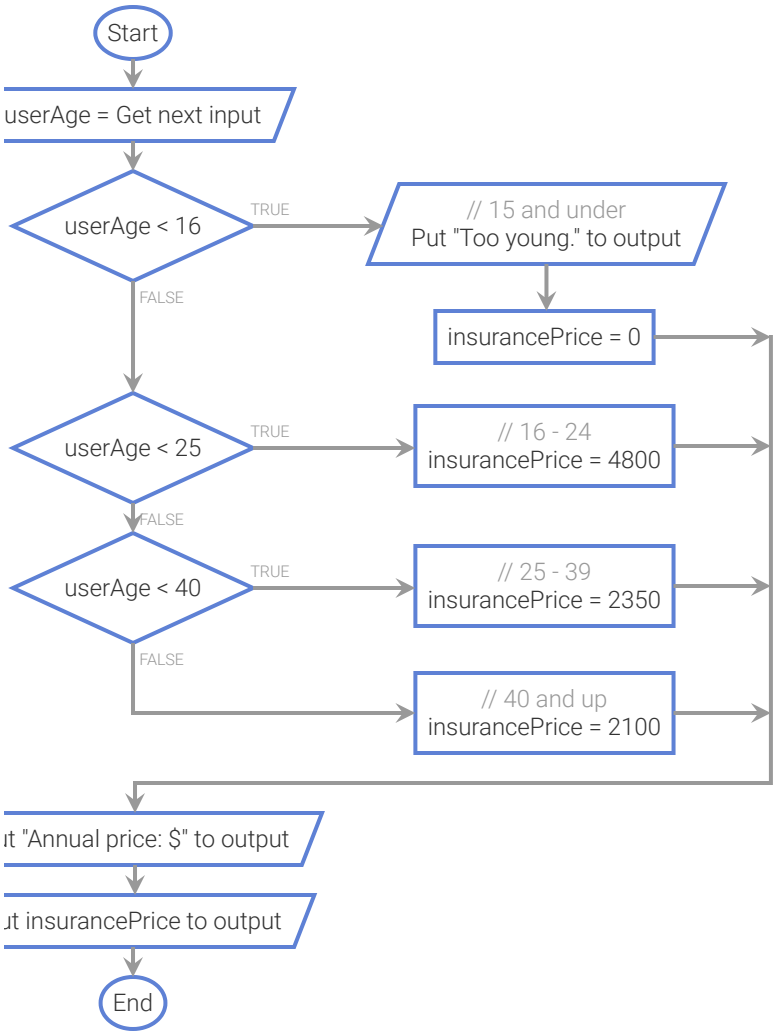
Check Next

Using multi-branch if-else to detect ranges

The sequential nature of multi-branch if-else statements is useful to detect ranges of numbers. In the following example, the second branch expression is only reached if the first expression is False. So the second branch is taken if userAge < 16 *is False* (so 16 or greater) AND userAge is < 25, meaning userAge is between 16-24 (inclusive).

PARTICIPATION ACTIVITY 4.3.4: Using if-elseif for ranges: Insurance prices.

Full screen



Variables

0	userAge inte
0	insurancePrice inte

Input

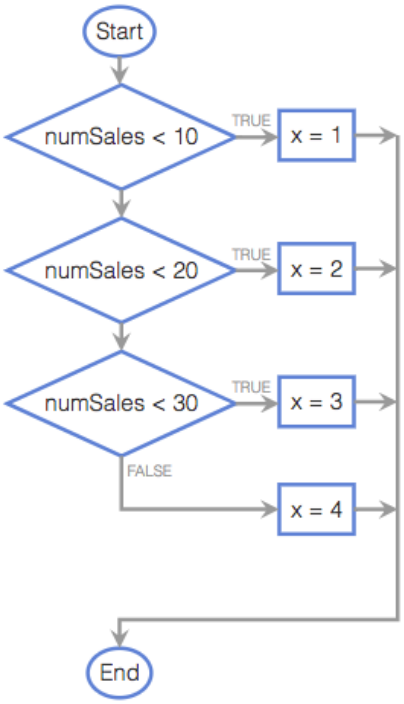
22

Output

NTER EXECUTION STEP RUN Execution speed Medium

PARTICIPATION ACTIVITY 4.3.5: Decision sequences and ranges.

Type the range for each branch. Type ranges as 25-29, or as 30+ for 30 and up.



1) Range for x = 2

Check Show answer

2) Range for x = 3

Check Show answer

3) Range for x = 4

Check Show answer

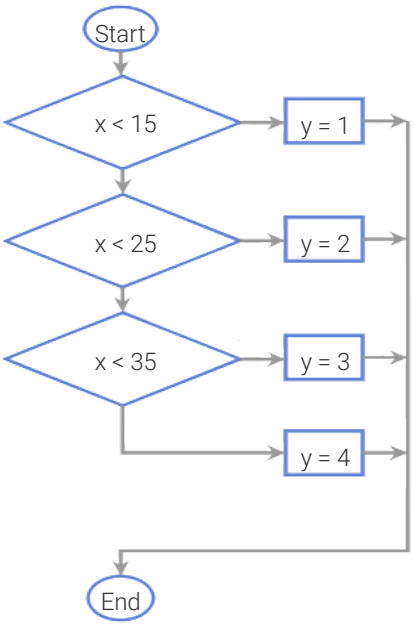
CHALLENGE ACTIVITY 4.3.2: Flowchart decision sequence to detect increasing ranges.

566436.4601014.qx3zqy7

Start

Type the range for the given branch. Assume x is a nonnegative integer.

Range for y = 1: Ex: 5 -



1 2 3 4

Check Next

4.4 Detecting ranges with branches

Relational operators

A ***relational operator*** checks how one operand's value relates to another, such as being greater than.

Some operators, such as >=, involve two characters. A programmer cannot arbitrarily combine the >, =, and < symbols; only the two-character sequences shown represent valid operators.

Table 4.4.1: Relational operators.

Relational operators	Description	Example (assume x is 3)
<	a **<** b means a is less than b.	x < 4 is True x < 3 is False

>	a **>** b means a is greater than b.	x > 2 is True x > 3 is False
<=	a **<=** b means a is less than or equal to b.	x <= 4 is True x <= 3 is True x <= 2 is False
>=	a **>=** b means a is greater than or equal to b	x >= 2 is True x >= 3 is True x >= 4 is False

PARTICIPATION ACTIVITY 4.4.1: Evaluating equations having relational operators.

Indicate whether the expression evaluates to True or False.
x is 5, y is 7.

1) x <= 7

- True
- False

2) y >= 7

- True
- False

3) Is x <> y a valid expression?

- Yes
- No

4) Is x =< y a valid expression?

- Yes
- No

PARTICIPATION ACTIVITY 4.4.2: Creating expressions with relational operators.

Type the operator to complete the desired expression.

1) num_dogs is greater than 10.

num_dogs [ ] 10

Check Show answer

2) num_cars is greater than or equal to 5.

num_cars [ ] 5

Check Show answer

3) num_cars is 5 or greater.

num_cars [ ] 5

Check Show answer

4) cents_lost is a negative number.

cents_lost [ ] 0

Check Show answer

Detecting ranges with if-else statements

Programmers commonly use the sequential nature of the multi-branch if-else arrangement to detect ranges of numbers. In the following example, the second branch expression is only reached if the first expression is false. So the second branch is taken if user_age < 16 *is false* (so 16 or greater) *and* user_age is < 25, meaning user_age is between 16 and 24 (inclusive).

PARTICIPATION ACTIVITY 4.4.3: Using the sequential nature of multi-branch if-else for ranges: Insurance prices.


```
user_age = int(input('Enter your age: '))

if user_age < 16:           # Age 15 and under
  print('Too young.')
  insurance_price = 0

elif user_age < 25:         # Age 16 - 24
  insurance_price = 4800

elif user_age < 40:          # Age 25 - 39
```

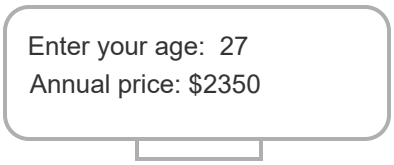
Memory

95		
96	27	user_age
97	2350	insurance_price
98		

```
  insurance_price = 2350

else:                          # Age 40 and up
  insurance_price = 2100

print(f'Annual price: ${insurance_price}')
```



27 < 16 X

27 < 25 X

27 < 40 ✓

Animation content:

The program shown:

```
user_age = int(input('Enter your age: '))

if user_age < 16:          # Age 15 and under
  print('Too young.')
  insurance_price = 0

elif user_age < 25:        # Age 16 - 24
  insurance_price = 4800

elif user_age < 40:        # Age 25 - 39
  insurance_price = 2350

else:                      # Age 40 and up
  insurance_price = 2100

print(f'Annual price: ${insurance_price}')
```

The console input/output shown:

```
Enter your age: 27
Annual price: $2350
```

Animation captions:

1. The user enters 27 for their age, which is stored in memory as the variable user_age. The multi-branch if-else first checks if user_age is less than 16, which is False.
2. The next branch in the multi-branch if-else checks if user_age is less than 25, which is False.
3. The next branch checks if user_age is less than 40, which is True. The elif's statements execute and the variable insurance_price is set to 2350 in memory.

PARTICIPATION ACTIVITY 4.4.4: Ranges and multi-branch if-else.

Type the range for each branch. Type ranges as: 25-29, or type 30+ for all numbers 30 and larger.

```
if num_sales < 10:
    ...
elif num_sales < 20:   # 2nd branch range: _____
    ...
elif num_sales < 30:   # 3rd branch range: _____
    ...
else:                  # 4th branch range: _____
    ...
```


1) 2nd branch range:

Check Show answer

2) 3rd branch range:

Check Show answer

3) 4th branch range:

Check Show answer

4) What is the range for the last branch below?

```
if num_items < 0:
    ...
elif num_items > 100:
    ...
else:    # Range: _______
    ...
```


Check Show answer

PARTICIPATION ACTIVITY

4.4.5: Complete the multi-branch if-else.

1) Second branch: user_num is less than 200

```
if user_num < 100 :
    ...
elif [          ] :
    ...
else :  # user_num >= 200
    ...
```

Check Show answer

2) Second branch: user_num is positive. (non-zero)

```
if user_num < 0 :
    ...
[          ] :
    ...
else :  # user_num is 0
    ...
```

Check Show answer

3) The second branch: user_num is greater than 105.

```
if user_num < 100 :
    ...
[                ] :
    ...
else :  # user_num is between
     # 100 and 105
    ...
```


Check Show answer

4) If the final else branch executes, what must user_num have been? Type "unknown" if appropriate.

```
if user_num <= 9:
    ...
elif user_num >= 11:
    ..
else:
    ... # user_num if this
executes?
```

Check Show answer

5) Which branch will execute? Valid answers: 1, 2, 3, or none.

```
user_num = 555;

if user_num < 0:
    ... # Branch 1
elif user_num > 666:
    ... # Branch 2
elif user_num < 100:
    ... # Branch 3
```


Check Show answer

CHALLENGE ACTIVITY 4.4.1: Detect ranges using branches.

Start

Type the program's output

```
x = int(input())

if x < 10:
    print('n')
else:
    print('t')

print('y')
```

Input

8

Output

```
n
y
```

1 2 3 4

Check Next

Operator chaining

Python supports ***operator chaining***. For example, `a < b < c` determines whether b is greater-than a but less-than c. Chaining performs comparisons left to right, evaluating a < b first. If the result is True, then b < c is evaluated next. If the result of the first comparison a < b is False, then there is no need to continue evaluating the rest of the expression. Note that a is not compared to c.

PARTICIPATION ACTIVITY 4.4.6: Chaining relational operators.

Write a relational expression using operator chaining.

1) x is less than y but greater than z.

Check Show answer

2) x is a nonnegative number less than 100.

```
if :
    # evaluated to True
else:
    # evaluated to False
```

Check Show answer

CHALLENGE ACTIVITY 4.4.2: If-else expression: Operator chaining.

Variable user_grade is read from input. Use operator chaining to complete the if-else expression as follows:

- If the value of user grade is between 9 and 12 (both inclusive), then "in high school" is output.
- Otherwise, "not in high school" is output.

Ex 1: If the input is 10, then the output is:

```
in high school
```

Ex 2: If the input is 8, then the output is:

```
not in high school
```

Note: 0 < x < 5 evaluates to true if x is between the ranges 0 and 5 (both non-inclusive).

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 user_grade = int(input())
2
3 if ''' Your solution goes here ''':
4     print('in high school')
5 else:
6     print('not in high school')
```


Run

View your last submission

CHALLENGE ACTIVITY 4.4.3: Basic if-else expressions.

566436.4601014.qx3zqy7

Start

Complete the if-else statement to output 'Fewer than 20' if the value of input_val is fewer than 20. Oth

Ex: If the input is `18`, then the output is:

```
Fewer than 20
```



```
input_val = int(input())

if ''' Your code goes here ''':
    print('Fewer than 20')
else:
    print('20 or more')
```

1 2

Check Next level

CHALLENGE ACTIVITY 4.4.4: Working with branches.

566436.4601014.qx3zqy7

Start

Integer number_of_pets is read from input. If number_of_pets is greater than 15, then output number ' is way too many pets.'

▶ **Click here for examples**

```
number_of_pets = int(input())

''' Your code goes here '''
```

```
4
```


1 2 3

Check Next level

CHALLENGE ACTIVITY 4.4.5: Working with ranges.

566436.4601014.qx3zqy7

Start

When the input variable light_freq is:

- less than or equal to 401, output 'Too low'.
- between 401 exclusive and 785 inclusive, output 'Can be seen'.
- greater than 785, output 'Too high'.

▶ **Click here for examples**

```
1 light_freq = int(input())
2
3 ''' Your code goes here '''
4
```

1 2

Check Next level

4.5 Detecting ranges using logical operators

Logical AND, OR, and NOT (general)

A ***logical operator*** treats operands as being True or False, and evaluates to True or False. Logical operators include AND, OR, and NOT. Programming languages typically use various symbols for those operators, but below the words AND, OR, and NOT are used for introductory purposes.

PARTICIPATION ACTIVITY 4.5.1: Logical operators: AND, OR, and NOT.

a	b	a AND b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a OR b
False	False	False
False	True	True
True	False	True
True	True	True

a	NOT a
False	True
True	False

Let x = 7, y = 9

(x > 0) AND (y < 10) True
True True

(x < 0) OR (y > 10) False
False False

NOT (x < 0) True
False

(x > 0) AND (y < 5) False
True False

(x < 0) OR (y > 5) True
False True

NOT (x > 0) False
True

Animation content:

Static Figure: Truth tables are shown for each of the logical operators and, or, and not. Examples of expressions using each logical operator are shown with each truth table.
Step 1: AND evaluates to True only if BOTH operands are True. The truth table for the logical operator and shows every permutation of possible values for a and b with the value of a AND b for

each permutation. The table is as follows:

a	b	a AND b
False	False	False
False	True	False
True	False	False
True	ztrue	True

The bottom row, where a is True, b is True, and a AND b is True, is highlighted. This is the only permutation where a AND b is True.

Step 2: OR evaluates to True if ANY operand is True (one, the other, or both). The truth table for the logical operator or is as follows:

a	b	a OR b
False	False	False
False	True	True
True	False	True
True	ztrue	True

All instances of True in the table are highlighted. In the second row, the value of b and the value of a OR b are both True. In the third row, the value of a and the value of a OR b are both True. In the third row, the value of a, the value of b, and the value or a OR b are all True. a OR b is only False in the first row, where the value of a and the value of b are both False.

Step 3: NOT evaluates to the opposite of the operand. The truth table for the logical operator not is as follows:

a	NOT a
False	True
True	False

Step 4: Each operand is an expression . If x = 7, y = 9, then (x greater than 0) AND (y less than 10) is True AND True, so the full expression evaluates to True (both operands are True). (x greater than 0) AND (y less than 5) is True and False, so the expression evaluates to False. (x less than 0) OR (y greater than 10) is False or False, so the expression evaluates to False. (x less than 0) OR (y greater than 5) is False or True, so the expression evaluates to True. NOT (x less than 0) is not False, so the

expression evaluates to True. NOT (x greater than 0) is not True, so the expression evaluates to False.

Animation captions:

1. AND evaluates to True only if BOTH operands are True.
2. OR evaluates to True if ANY operand is True (one, the other, or both).
3. NOT evaluates to the opposite of the operand.
4. Each operand is an expression. If x = 7, y = 9, then (x > 0) AND (y < 10) is True AND True, so the full expression evaluates to True (both operands are True).

Table 4.5.1: Logical operators.

Logical operator	Description
a AND b	***Logical AND***: True when both of its operands are True.
a OR b	***Logical OR***: True when at least one of its two operands are True.
NOT a	***Logical NOT***: True when its one operand is False, and vice versa.

PARTICIPATION ACTIVITY 4.5.2: Evaluating expressions with logical operators.

Indicate whether the expression evaluates to True or False.
x is 7, y is 9.

1) x > 5

- True
- False

2) (x > 5) AND (y < 20)

- True
- False

3) (x > 10) AND (y < 20)

- True
- False

4) (x > 10) OR (y < 20)

- True
- False

5) (x > 10) OR (y > 20)

- True
- False

6) NOT (x > 10)

- True
- False

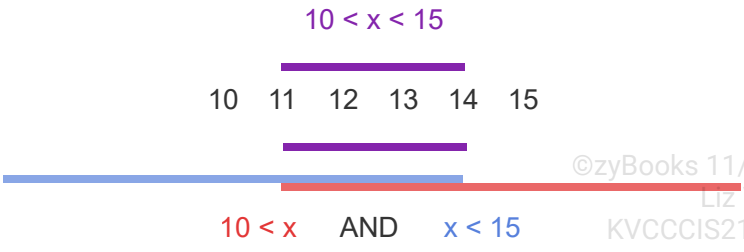
7) NOT ((x > 5) AND (y < 20))

- True
- False

Detecting ranges with logical operators (general)

A common use of logical operators is to detect if a value is within a range.

PARTICIPATION ACTIVITY 4.5.3: Using AND to detect if a value is within a range.



Animation content:

Static figure: A number line is shown with the values 10 through 15 labeled.

Animation captions:

1. The range 10 < x < 15 means that x may be 11, 12, 13, or 14.
2. Specifying that range in a program can be done using two < operators along with an AND operator. 10 < x defines the range 11 and higher.
3. x < 15 defines the range 14 and lower. ANDing yields the overlapping range. Only when x is 11, 12, 13, or 14 will both expressions be true.

PARTICIPATION ACTIVITY 4.5.4: Using AND to detect if a value is within a range.

Assume x is an integer.

1) Which approach uses a logical operator to detect if x is in the range 1 to 99?
   - 0 < x < 100
   - (0 < x) AND (x < 100)
   - (0 < x) AND (x > 100)

2) Which approach detects if x is in the range -4 to +4?
   - (x < -5) AND (x < 5)
   - (x > -5) OR (x < 5)
   - (x > -5) AND (x < 5)

3) Which detects if x is either less than -5, or greater than 10?
   - (x < -5) AND (x > 10)
   - (x < -5) OR (x > 10)

Booleans and logical operators

A ***Boolean*** refers to a value that is either True or False. Note that True and False are keywords in Python and must be capitalized. A programmer can assign a Boolean value by specifying True or False, or by evaluating an expression that yields a Boolean.

Figure 4.5.1: Creating a Boolean.

```
my_bool = True    # Assigns my_bool with the boolean value True

is_small = 4 < 3  # Assigns is_small with the result of the expression 4 < 3
(False)
```


Keywords ***and***, ***or***, and ***not*** (lowercase) are used to represent the AND, OR, and NOT logical operators. Logical operators are commonly used in expressions of if-else statements.

Table 4.5.2: Logical operators.

Logical operator	Description
a **and** b	***Boolean AND***: True when both operands are True.
a **or** b	***Boolean OR***: True when at least one operand is True.
not a	***Boolean NOT*** (opposite): True when the single operand is False (and False when operand is True).

Table 4.5.3: Logical operators examples.

Given age = 19, days = 7, user_char = 'q'

`(age > 16) and (age < 25)`	True, because both operands are True.
`(age > 16) and (days > 10)`	False, because both operands are not True (days > 10 is False).
`(age > 16) or (days > 10)`	True, because at least one operand is True (age > 16 is True).
`not (days > 10)`	True, because operand is False.
`not (age > 16)`	False, because operand is True.
`not (user_char == 'q')`	False, because operand is True.

PARTICIPATION ACTIVITY 4.5.5: Logical operators: Complete the expressions to detect the desired range.

1) days_logged is greater than 30 and less than 90

```
if (days_logged > 30)
(days_logged < 90):
```

Check Show answer

2) 0 < max_cars < 100

```
if (max_cars > 0)
(max_cars < 100):
```

Check Show answer

3) num_stores is between 10 and 20, inclusive.

```
if (num_stores >= 10)
(num_stores <= 20):
```

Check Show answer

4) not_valid is either less than 15 or greater than 79.

```
if (not_valid < 15)
(not_valid > 79):
```

Check Show answer

PARTICIPATION ACTIVITY 4.5.6: Creating expressions with logical operators.

1) num_dogs has a minimum of 2 and a maximum of 5.

```
if (num_dogs >= 2)
    :
```

Check Show answer

2) wage is greater than 10 and less than 18. Use > and < (not >= and <=). Use parentheses around sub-expressions.


```
if    :
```

Check Show answer

3) num is a 3-digit positive integer. Ex: 100, 989, and 523, are 3-digit positive integers, but 55, 1000, and -4 are not.

For most direct readability, your expression should compare directly with the smallest and largest 3-digit number.

```
if (num >= 100)
    :
```

Check Show answer

Example: TV channels

A cable TV provider has regular channels numbered 2-499, and high-definition channels numbered 1002-1499. A program sets a character variable to 's' for standard, 'h' for high-definition, and 'e' for error.

Figure 4.5.2: Detecting ranges: Cable TV channels.

```
if (user_channel >= 2) and (user_channel <= 499):
   channel_type = 's'

elif (user_channel >= 1002) and (user_channel <=
1499):
   channel_type = 'h'

else:
   channel_type = 'e'
```


zyDE 4.5.1: Detecting ranges: Cable TV channels.

Run the program and observe the output. Change the input box value from 3 to another number and run again.

Load default template...

```
1  user_channel = int(input())
2
3  if (user_channel >= 2) and (
4      channel_type = 's'
5
6  elif (user_channel >= 1002)
7      channel_type = 'h'
8
9  else:
10     channel_type = 'e'
11
12 print(f'Channel type: {chanr
13
```

3

Run

PARTICIPATION ACTIVITY 4.5.7: TV channel example: Detecting ranges.

Consider the above example.

1) If user_channel is 300, to what does the if statement's expression, `(user_channel >= 2) and (user_channel <= 499)`, evaluate?

- True
- False

2) If user_channel is 300, does the else if expression `(user_channel >= 1002) and (user_channel <= 1499)` get checked?

- Yes
- No

3) Did the expressions use logical AND or logical OR?

- AND
- OR

4) Channels 500-599 are pay channels. Does this expression detect that range?
`(user_channel >= 500) or (user_channel <= 599)`

- Yes
- No

Detecting ranges implicitly vs. explicitly

A programmer often uses logical operators to detect a range by explicitly specifying the high end and low end of the range. However, if a program should detect increasing ranges without gaps, a multi-branch if-else statement can be used without logical operators; the low end of the range is implicitly known upon reaching an expression. Likewise, a decreasing range without gaps has implicitly known high-ends.

PARTICIPATION ACTIVITY 4.5.8: Detecting ranges implicitly vs. explicitly.


```
if x < 0 :
    # Negative

elif  (x >= 0) and (x <= 10) :
    # 0..10

elif  (x >= 11) and (x <= 20) :
    # 11..20
```

```
if x < 0 :
  # Negative

elif (x <= 10) :
  # 0..10

elif (x <= 20) :
  # 11..20
```

x >= 0 is implicit

x > 10 is implicit

x > 20 is implicit

```
    else :
        # 21+
```

Explictly defined ranges

```
else :
  # 21+
```

Implictly defined ranges

Animation content:

Static figure: Begin Python code:
if x < 0 :
  # Negative

elif (x >= 0) and (x <= 10) :
  # 0..10

elif (x >= 11) and (x <= 20) :
  # 11..20

else :
  # 21+
End Python code. This code block is labeled "Explicitly defined ranges." Begin Python code:
if x < 0 :
 # Negative

elif (x <= 10) :
 # 0..10

elif (x <= 20) :
 # 11..20

else :
 # 21+
End Python code. This code block is labeled "Implicitly defined ranges."

Step 1:
Begin Python code:
if x < 0 :
  # Negative

elif (x >= 0) and (x <= 10) :
  # 0..10

elif (x >= 11) and (x <= 20) :
  # 11..20

```
else :
   # 21+
```

End Python code. This code block is labeled "Explicitly defined ranges".

Step 2: If the first branch doesn't execute, x must be >= 0. So the second branch's expression is x <= 10. The x >= 0 is implicit. Begin Python code:

```
if x < 0 :
  # Negative

elif (x <= 10) :
  # 0..10

elif (x <= 20) :
  # 11..20

else :
  # 21+
```


End Python code. This code block is labeled "Implicitly defined ranges." In the "explicitly defined ranges" code block, x >= 0 is underlined, and x <= 10 is underlined. In the "implicitly defined ranges" code block, x <= 10 is underlined. the text "x >= 0 is implicit" appears.

Step 3: Implicit ranges can simplify a multi-branch if the statement for ranges is without gaps. In the "explicitly defined ranges" code block, x >= 11 is underlined, and x <= 20 is underlined. In the "implicitly defined ranges" code block, x <= 20 is underlined. The text "x > 10 is implicit" appears. The text "x > 20 is implicit" appears next to the else statement.

Animation captions:

1. This code detects ranges explicitly using the AND operator. The first branch executes when x < 0; the second executes when (x >=0) and (x <= 10).
2. If the first branch doesn't execute, x must be >= 0. So the second branch's expression is x <= 10. The x >= 0 is implicit.
3. Implicit ranges can simplify a multi-branch if the statement for ranges is without gaps.

PARTICIPATION ACTIVITY 4.5.9: Detecting ranges implicitly vs explicitly.

For each pair of statements, does the second if-else statement detect the same ranges as the first if-else statement?

1)

```
if temp <= 0...
elif (temp > 0) and (temp <
100)...
```

```
if temp <= 0...
elif temp < 100...
```

- Yes
- No

2)


```
if systolic < 130: ...
elif (systolic >= 130) and
(systolic <= 139): ...
```

```
if systolic < 130: ...
elif systolic >= 130: ...
```

- Yes
- No

3)

```
if (year >= 1901) and (year <=
2000): ...
elif (year >= 2001) and (year
<= 2100): ...
```

```
if year <= 2000: ...
elif year <= 2100: ...
```

- Yes
- No

CHALLENGE ACTIVITY 4.5.1: Detecting ranges using logical operators.

566436.4601014.qx3zqy7

Start

Modify the given if statement so that 'Large university' is output if enrollment_input is in the range 85

▶ **Click here for examples**


```
1 enrollment_input = int(input())
2
3 # Modify the following line
4 if (enrollment_input < 8500) and (enrollment_input > 18000):
5     print('Large university')
6 else:
7     print('Not a large university')
```

1 2 3

Check Next level

4.6 Detecting ranges with gaps

Basic ranges with gaps

Ranges contain gaps. Ex: Movie theaters give ticket discounts to children (anyone 12 and under) and seniors (anyone 65 and older). The gap is the group of people aged 13 to 64. An if-else statement can be used to detect such ranges with gaps.

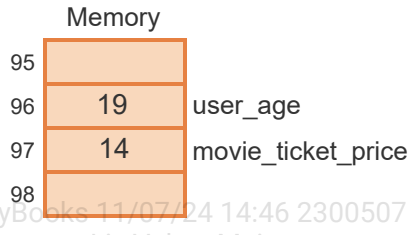
PARTICIPATION ACTIVITY 4.6.1: Using multi-branch if-else for detecting ranges with gaps: Movie ticket prices.

```
movie_ticket_price = None

user_age = int(input('Enter your age: '))

if user_age <= 12:      # Age 12 and under
   print('Child ticket discount.')
   movie_ticket_price = 11
elif user_age >= 65:    # Age 65 and older
   print('Senior ticket discount.')
   movie_ticket_price = 12
else:                   # All other ages
   movie_ticket_price = 14

print(f'Movie ticket price: ${movie_ticket_price}')
```




```
Enter your age:  67
Senior ticket discount.
Movie ticket price: $12

Enter your age:  19
Movie ticket price: $14
```

67 <= 12 X
67 >= 65 ✓

19 <= 12 X
19 >= 65 X

Animation content:

Static figure: Begin Python code: movie_ticket_price = None

```
user_age = int(input('Enter your age: '))

if user_age <= 12:     # Age 12 and under
   print('Child ticket discount.')
   movie_ticket_price = 11
elif user_age >= 65:   # Age 65 and older
   print('Senior ticket discount.')
   movie_ticket_price = 12
else:                  # All other ages
   movie_ticket_price = 14

print(f'Movie ticket price: ${movie_ticket_price}')
```

End Python code. A memory block stores the variable user_age with value 19 in address 96. The variable movie_ticket_price with value 14 is stored in address 97. The console contains 6 lines of output:
Enter your age: 67
Senior ticket discount.
Movie ticket price: $12

Enter your age: 19
Movie ticket price: $14
Comparisons with a true or false are shown. 67 <= 12, false. 67 >= 65, pass. 19 <= 12, false. 19 >= 65, false.
Step 1: After the user enters their age, the else-if branch's first branch checks if age is <= 12. The line of code user_age = int(input('Enter your age: ')) is highlighted. "Enter your age: 67" is outputted. user_age is assigned with the value 67 and stored in memory at address 96.
Step 2: user_age is 67, which is greater than 12, so the program moves to the second branch that checks if user_age is >= 65. 67 <= 12, false. The line of code elif user_age >= 65: # Age 65 and older is highlighted.
Step 3: 67 is >= 65, so the second branch's statements execute, applying the senior discount to the ticket price. The program concludes by outputting the ticket price. 67 >= 65, true. The line of code print('Senior ticket discount.') is highlighted. "Senior ticket discount." is outputted. The line of code movie_ticket_price = 12 is highlighted. movie_ticket_price is assigned with the value 12 and stored in memory at address 97. The line of code print(f'Movie ticket price: ${movie_ticket_price}') is

highlighted. "Movie ticket price: $12" is outputted.
Step 4: If the user's age falls between the gap of 12 and 65 (13 to 64), the else branch executes and the ticket price is $14, the most expensive price. The line of code user_age = int(input('Enter your age: ')) is highlighted. "Enter your age: 19" is outputted. user_age is assigned with the value 19 and stored in memory at address 96. The line of code if user_age <= 12: is highlighted. 19 <=12, false. The line of code elif user_age >= 65: is highlighted. 19 >= 65, false. The following lines of code are highlighted:

```
else:               # All other ages
  movie_ticket_price = 14
```


movie_ticket_price is assigned with the value 14 and stored in memory at address 97. The line of code print(f'Movie ticket price: ${movie_ticket_price}') is highlighted. "Movie ticket price: $14" is outputted.

Animation captions:

1. After the user enters their age, the else-if branch's first branch checks if age is <= 12.
2. user_age is 67, which is greater than 12, so the program moves to the second branch that checks if user_age is >= 65.
3. 67 is >= 65, so the second branch's statements execute, applying the senior discount to the ticket price. The program concludes by outputting the ticket price.
4. If the user's age falls between the gap of 12 and 65 (13 to 64), the else branch executes and the ticket price is $14, the most expensive price.

PARTICIPATION ACTIVITY 4.6.2: Detecting ranges with gaps and multi-branch if-else.

Select the correct answers below.

1) In the animation above, what is the age range for a child ticket discount?
   - 0-12
   - less than 13
   - less than 11

2) In the animation above, what is the age range for a senior ticket discount?
   - 65 or more
   - 66 or more
   - 13-64

3) What is the range for the last branch below?

```
if num_items <= 0:
    ...
elif num_items > 100:
    ...
else:  # Range: _______
    ...
```

- 1-99
- 0-100
- 1-100

4) What is the range for the last branch below?

```
if num_items < 50:
    ...
elif num_items > 50:
    ...
else:  # Range: _______
    ...
```

- 49-51
- 0-50
- 50

Ranges with gaps using logical operators

Programmers often use logical operators to explicitly detect ranges with an upper and lower bound, including ranges with gaps that may have intermediate bounds. Ex: If a valid office number is within the ranges of 100 to 150 or 200 to 250, the logical AND operator or operator chaining can be used to identify the lower and upper bounds of the two ranges. Further, the ranges can be combined into a single branch using the logical OR operator.

PARTICIPATION ACTIVITY 4.6.3: Explicit ranges with gaps detection using logical AND and OR.


```
if office_num >= 100 and office_num <= 150:
    # valid office number

elif office_num >= 200 and office_num <= 250:
    # valid office number

else:
    # invalid office number
```

```
if (office_num >= 100 and office_num <= 150) or (office_num >= 200 and office_num <= 250):
    # valid office number

else:
    # invalid office number
```


Animation content:

Static figure: A code block is displayed.
Begin Python code:
if office_num >= 100 and office_num <= 150:
    # valid office number

elif office_num >= 200 and office_num <= 250:
    # valid office number

else:
    # invalid office number
End Python code.
Step 1: The logical AND operator is used to identify the lower and upper bounds of the two valid ranges of office numbers (100 to 150 and 200 to 250). Any number outside of the ranges is in the gap. The lines of code, if office_num >= 100 and office_num <= 150:, # valid office number, elif office_num >= 200 and office_num <= 250:, # valid office number, are highlighted.
Step 2: Further, the two ranges can be combined into a single branch using the logical OR operator. A new code block is displayed.
Begin Python code:
if (office_num >= 100 and office_num <= 150) or (office_num >= 200 and office_num <= 250):
    # valid office number

else:
    # invalid office number
End Python code.
The lines of code, if (office_num >= 100 and office_num <= 150) or (office_num >= 200 and office_num <= 250):, # valid office number, are highlighted.

Animation captions:

1. The logical AND operator is used to identify the lower and upper bounds of the two valid ranges of office numbers (100 to 150 and 200 to 250). Any number outside of the ranges is i the gap.
2. Further, the two ranges can be combined into a single branch using the logical OR operator.

PARTICIPATION ACTIVITY 4.6.4: NFL Jersey numbers.

In the National Football League (NFL), player positions have jersey numbers in specific ranges. Ex: An NFL wide receiver can only wear jersey numbers from 10 to 19 or 80 to 89.

Select the if statement that explicitly detects the correct NFL jersey number ranges.

1) Linebacker: 40 to 59 or 90 to 99

- `if (j_num >= 40 and j_num <= 59) or (j_num >= 90 and j_num <= 99):`
- `if (j_num > 40 and j_num <= 59) or (j_num > 90 and j_num <= 99):`
- `if j_num >= 40 and j_num <= 99:`

2) Tight end: 40 to 49 or 80 to 89

- `if (40 <= j_num <= 49) and (80 <= j_num <= 89):`
- `if (j_num >= 40 or j_num <= 49) and (j_num >= 80 or j_num <= 89):`
- `if (40 <= j_num <= 49) or (80 <= j_num <= 89):`

3) Defensive lineman: 50 to 79 or 90 to 99

- `if (j_num > 50 and j_num < 79) or (j_num > 90 and j_num < 99):`
- `if (j_num >= 49 and j_num =< 80) or (j_num >= 89 and j_num <= 100):`
- `if (j_num > 49 and j_num < 80) or (j_num > 89 and j_num < 100):`

4) Quarterback: 1 to 19

- `if j_num <= 19:`
- `if j_num > 0 and j_num < 20:`

```
if j_num > 0 or j_num <
20:
```

CHALLENGE ACTIVITY 4.6.1: Enter the output of the branch expressions.

566436.4601014.qx3zqy7

Start

Type the program's output

```
x = 5

if (x < 7) and (x > 2):
    print('a')
else:
    print('b')
```

a

1 2 3 4

Check Next

CHALLENGE ACTIVITY 4.6.2: Ranges with gaps.

566436.4601014.qx3zqy7

Start

Integer lemon_count is read from input representing the number of lemons. Output 'Unsuitable batch

- the number of lemons is fewer than or equal to 10.
- or the number of lemons is more than 15.

▶ **Click here for example**

```
1 lemon_count = int(input())
2
3 ''' Your code goes here '''
4
```

1 2 3

Check Next level

4.7 Detecting multiple features with branches

Multiple distinct if statements

Sometimes the programmer has multiple if statements in sequence, which looks similar to a multi-branch if-else statement but has a very different meaning. Each if statement is independent and more than one branch can execute, in contrast to the multi-branch if-else arrangement.

The below Python Tutor tool traces a Python program's execution. The Python Tutor tool is available at www.pythontutor.com.

PythonTutor: Multiple distinct if statements.

```
1  user_age = 26  # Hardcoded for this tool. Could replace with "int(inp
2
3  # Note that more than one "if" statement can execute
4  if user_age < 16:
5      print('Enjoy your early years.')
6
7  if user_age > 15:
8      print('You are old enough to drive.')
9
10 if user_age > 17:
11     print('You are old enough to vote.')
12
13 if user_age > 24:
14     print('Most car rental companies will rent to you.')
15
16 if user_age > 34:
17     print('You can run for president.')
```


line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 1 of 9

Program output

PARTICIPATION ACTIVITY 4.7.1: If statements.

Determine the final value of num_boxes.

1)
```
num_boxes  = 0
num_apples = 9

if num_apples < 20:
    num_boxes = 3
if num_apples < 10:
    num_boxes = num_boxes -
1
```

Check Show answer

2)

```
num_boxes  = 0
num_apples = 12

if num_apples < 4:
    num_boxes = 1
elif num_apples < 14:
    num_boxes = num_boxes +
3
else:
    num_boxes = num_boxes +
5
```


Check Show answer

CHALLENGE ACTIVITY 4.7.1: Enter the output for the multiple if-else branches.

566436.4601014.qx3zqy7

Start

Type the program's output

```
num_items = 5

if num_items < 1:
   print('a')
elif num_items < 8:
   print('e')
else:
   print('h')

print('p')
```

ep

1 2 3

Check Next

Nested if-else statements

A branch's statements can include any valid statements, including another if-else statement, which are known as ***nested if-else*** statements.

PythonTutor: Nested if-else

```
1  user_choice = 2  # Hardcoded values for this tool. Could be
2  num_items = 5
3
4  if user_choice == 1:
5     print('user_choice is 1')
6  elif user_choice == 2:
7     if num_items < 0:
8        print('user_choice is 2 and num_items < 0')
9     else:
10       print('user_choice is 2 and num_items >= 0')
11 else:
12    print('user_choice is neither 1 or 2')
```


line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 1 of 6

Program output

PARTICIPATION ACTIVITY 4.7.2: Nested if-else statements.

Determine the final value of sales_bonus given the initial values specified below.

```
if sales_type == 2:
    if sales_bonus < 5:
        sales_bonus = 10
    else:
        sales_bonus = sales_bonus + 2
else:
    sales_bonus = sales_bonus + 1
```


1) sales_type = 1; sales_bonus = 0;

- 0
- 1

- 10

2) sales_type = 2; sales_bonus = 4;

- 5
- 6
- 10

3) sales_type = 2; sales_bonus = 7;

- 8
- 9
- 10

CHALLENGE ACTIVITY 4.7.2: Detecting multiple features with branches.

566436.4601014.qx3zqy7

Start

allowance_earned is read from input. Write multiple if statements:

- If allowance_earned is greater than 6, then output 'Buy a keychain.'
- If allowance_earned is greater than or equal to 26, then output 'Buy a new coat.'
- If allowance_earned is less than or equal to 2, then output 'Save money.'

▶ **Click here for examples**

```
allowance_earned = int(input())

''' Your code goes here '''

```


1 2

Check Next level

4.8 Comparing data types and common errors

Comparing integers, strings, and floating-point types

The relational and equality operators work for integer, string, and floating-point built-in types.

Floating-point types should not be compared using the equality operators, due to the imprecise representation of floating-point numbers.

The operators can also be used for the string type. Strings are equal if they have the same number of characters and corresponding characters are identical. If string my_str = 'Tuesday', then (my_str == 'Tuesday') is True, while (my_str == 'tuesday') is False because T differs from t.

PARTICIPATION ACTIVITY 4.8.1: Comparing various types.

Which comparisons will not result in a syntax error AND consistently yield expected results? Variables have types denoted by their names.

1) my_int == 42
   - OK
   - Not OK

2) my_float == 3.14
   - OK
   - Not OK

3) my_string == 'Hello'
   - OK
   - Not OK

The types of the values being compared determines the meaning of a comparison. If both values are numbers, then the numbers are compared arithmetically (`5 < 2` is False). Comparisons that make no sense, such as `1 < 'abc'`, result in a TypeError.

Comparison of values with the same type, such as `5 < 2`, or `'abc' >= 'ABCDEF'`, depends on the types being compared.

- Numbers are arithmetically compared.
- Strings are compared by converting each character to a number value (ASCII or Unicode), and then comparing each character in order. Most string comparisons use equality operators "==" or "!=", as in `today == 'Friday'`.
- Lists and tuples are compared via an ordered comparison of every element in the sequence. Every element between the sequences must compare as equal for an equality operator to evaluate to True. Relational operators like < or > can also be used: The result is determined by the first mismatching elements in the sequences. For example, if x = [1, 5, 2] and y = [1, 4, 3], then evaluating `x < y` first evaluates that 1 and 1 match. Since the first list elements match, neither list can be considered to be less than the other, nor can the lists be declared equal without comparing more elements. So the next elements must be compared. The next elements do not match, so `5 < 4` is evaluated, which produces a value of False.
- Dictionaries are compared only with == and !=. To be equal, two dictionaries must have the same set of keys and the same corresponding value for each key.

PARTICIPATION ACTIVITY 4.8.2: Comparing various types.

1) Click the expression that is False.
   - 5 <= 5.0
   - 10 != 9.999999
   - (4 + 1) != 5.0
2) Click the expression that is False.
   - 'FRIDAY' == 'friday'
   - '1' < '2'
   - 'a' != 'b' < 'c'
3) Click the expression that is True.
   - {'Henrik': '$25'} == {'Daniel': '$25'}
   - (1,2,3) > (0,2,3)
   - [1, 2, 3] >= ['1', '2', '3']

Common branching errors

A common error is to use = rather than == in an if-else expression, as in: `if numDogs = 9:`. In such cases, the interpreter should generate a syntax error.

Another common error is to use invalid character sequences like =>, !<, or <>, which are not valid operators.

PARTICIPATION ACTIVITY 4.8.3: Watch out for assignment in an if-else expression.

What is the final value of num_items? Write "Error" if the code results in an error.

1)
```
num_items = 3
if num_items == 3:
   num_items = num_items + 1
```

Check Show answer

2)
```
num_items = 3
if num_items = 10:
   num_items = num_items + 1
```

Check Show answer

3)
```
num_items = 3
if num_items > 10:
   num_items = num_items + 1
```

Check Show answer

CHALLENGE ACTIVITY 4.8.1: If-else statement: Fix errors.

566436.4601014.qx3zqy7

Start

Find and fix the error in the if-else statement.

```
1 user_num = int(input())     # Program will be tested with values: 1, 2, 3, 0.
2
3 if user_num = 2:
4     print('Num is equal to two')
5 else:
6     print('Num is not two')
7
```

8

1

Check Try again

4.9 Membership and identity operators

Membership operators: in/not in

One programming task involves determining whether a specific value can be found within a container, such as a list or dictionary. The ***in*** and ***not in*** operators, known as ***membership operators***, yield True or False if the left operand matches the value of an element in the right operand, which is always a container.

PARTICIPATION ACTIVITY 4.9.1: Membership operators: Checking for a value in a list.

Animation content:

Code is written to create a list called prices with 3 elements: prices = ['$20', 15, 5]. A line of code is written: print (15 in prices). 15 is compared to all elements in the list. 15 is not in the first location ('$20' is there). 15 is in the second location. True is returned and printed to output. Then a line of code is written print (44 in prices). 44 is compared to all elements in the list. 44 is not in any location of the list. False is returned and printed to output.

Animation captions:

1. A user creates a new list.
2. Every element in the list is checked for the value of 15.
3. Every element in the list is checked for the value of 44, which is not found. So the statement (44 in prices) evaluates to False.

The membership operators can be used with sequence types. If the variable x is a list or tuple, then `a in x` evaluates to True if there exists an index idx for which `a == x[idx]` is True. The program below demonstrates membership operator usage in a list:

Figure 4.9.1: Membership operators example: Checking for an item in a list.

```
# Use the "in" operator
barcelona_fc_roster = ['Alves',
'Messi', 'Fabregas']

name = input('Enter name to
check: ')

if name in barcelona_fc_roster:
    print(f'Found {name} on the
roster.')
else:
    print(f'Could not find
{name} on the roster.')
```

```
Enter name to check: Messi
Found Messi on the roster.
...
Enter name to check: Rooney
Could not find Rooney on the roster.
```

```
# Use the "not in" operator
barcelona_fc_roster = ['Alves',
'Messi', 'Fabregas']

name = input('Enter name to
check: ')

if name not in
barcelona_fc_roster:
    print(f'Could not find
{name} on the roster.')
else:
    print(f'Found {name} on the
roster.')
```

```
Enter name to check: Messi
Found Messi on the roster.
...
Enter name to check: Rooney
Could not find Rooney on the roster.
```


Membership operators can be used to check whether a string is a ***substring***, or matching subset of characters, of a larger string. For example, `'abc' in '123abcd'` returns True because the substring abc exists in the larger string.

Figure 4.9.2: Checking for substrings.

```
request_str = 'GET index.html
HTTP/1.1'

if '/1.1' in request_str:
    print('HTTP protocol 1.1')

if 'HTTPS' not in request_str:
    print('Unsecured connection')
```

```
HTTP protocol 1.1
Unsecured
connection
```


Membership in a dictionary (dict) implies that a specific *key* exists in the dictionary. *A common error is to assume that a membership operator checks the values of each dictionary key as well.*

Figure 4.9.3: Checking for membership in a dict.

```
my_dict = {'A': 1, 'B': 2, 'C': 3}

if 'B' in my_dict:
   print("Found 'B'")
else:
   print("'B' not found")

# Membership operator does not check
values
if 3 in my_dict:
    print('Found 3')
else:
    print('3 not found')
```

```
Found 'B'
3 not
found
```

PARTICIPATION ACTIVITY 4.9.2: Membership operators.

1) Which expression checks whether the list my_list contains the value 15?

- 15 in my_list[0]
- 15 in my_list
- my_list['15'] != 0

2) Which expression checks if the value 10 exists in the dictionary my_dict?

- 10 in my_dict['key']
- 10 in my_dict
- None of the above

Identity operators: is/is not

Sometimes a programmer wants to determine whether two variables are the same object. The programmer can use the ***identity operator***, ***is***, to check whether two operands are bound to a single object. The inverse identity operator, ***is not***, gives the negated value of "is". Thus, if `x is y` is True, then `x is not y` is False.

Identity operators do not compare object values; rather, identity operators compare object identities to determine equivalence. Object identity is usually the memory address of an object. Thus, identity operators return True only if the operands reference the same object.

A common error is to confuse the equivalence operator "==" and the identity operator "is" because a statement such as `if x is 3` is valid syntax and is grammatically appealing. Python may confusedly evaluate the statement `x is 3` as True, but `y is 1000` as False, when x = 3 and y = 1000. Python interpreters typically precreate objects for a small range of numbers to avoid constantly recreating objects for such small values. In the example above, an object for 3 was precreated, so x references the same object as the literal. However, Python did not precreate an object for 1000. *A good practice is to avoid using the identity operators "is" and "is not", unless explicitly testing whether two objects are identical.*

The id() function can be used to retrieve the identifier of any object. If `x is y` is True, then `id(x) == id(y)` is also True.

Figure 4.9.4: Identity operators.

```
w = 500
x = 500 + 500  # Create a new object with
value 1000
y = w + w      # Create a second object
with value 1000
z = x          # Bind z to the same object
as x

if z is x:
    print('z and x are bound to the same
object')
if z is not y:
    print('z and y are NOT bound to the
same object')
```

```
z and x are bound to the same
object
z and y are NOT bound to the
same object
```


PARTICIPATION ACTIVITY 4.9.3: Membership and identity operators.

Write the simplest expression that captures the desired comparison.

1) x is a key in the dict my_dict.

Check Show answer

2) The variables x and y are unique objects.

Check Show answer

3) The character 'G' exists in the string my_str.

Check Show answer

4) my_str is not the third element in the list my_list.

Check Show answer

CHALLENGE ACTIVITY 4.9.1: Membership and Identity: Enter the output of the code.

566436.4601014.qx3zqy7

Start

Type the program's output

```
numbers = [5, 6, 7, 8]
x = int(input())

if x in numbers:
    print('in')
else:
    print('not in')
```

Input

5

Output

```
in
```

1 2 3

Check Next

CHALLENGE ACTIVITY 4.9.2: Boolean operators: Detect specific values.

Write an expression using membership operators that prints "Special number" if special_num is one of the special numbers stored in the list special_list = [-99, 0, 44].

Sample output with input: 17

```
Not special number
```


Learn how our autograder works

566436.4601014.qx3zqy7

```
1 special_list = [-99, 0, 44]
2 special_num = int(input())
3
4 if ''' Your solution goes here ''':
5     print('Special number')
```

```
else:
    print('Not special number')
```


Run

View your last submission

Object identity

Object identity is an implementation detail of Python. For the standard CPython implementation, identity is the memory address of the object.

4.10 Order of evaluation

Precedence rules

The order in which operators are evaluated in an expression is known as ***precedence rules***. Arithmetic, logical, and relational operators are evaluated in the order shown below.

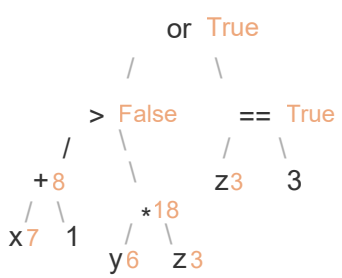
Table 4.10.1: Precedence rules for arithmetic, logical, and relational operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.

	evaluated first	
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
`< <= > >= == !=`	Relational, (in)equality, and membership operators	`x < 2 or x >= 10` is evaluated as `(x < 2) or (x >= 10)` because < and >= have precedence over or.
not	not (logical NOT)	`not x or y` is evaluated as `(not x) or y`.
and	Logical AND	`x == 5 or y == 10 and z != 10` is evaluated as `(x == 5) or ((y == 10) and (z != 10))` because and has precedence over or.
or	Logical OR	`x == 7 or x < 2` is evaluated as `(x == 7) or (x < 2)` because < and == have precedence over or.

PARTICIPATION ACTIVITY 4.10.1: Applying the precedence rules to an expression can be thought of as a tree.

x + 1 > y * z or z == 3



Animation content:

Static figure: The expression x + 1 > y * z or z == 3 is displayed, as text and as a tree.

Step 1: Expressions like x + 1 > y * z or z == 3 are evaluated using precedence rules. Among +, >, *, or, and ==, the * comes first.

Step 2: Next comes +, >, ==, and finally or.

Step 3: The expression is actually treated like a tree, evaluated from the bottom up.

Operator	Left Branch	Right Branch
or	>	==
==	z	3
>	+	*
+	x	1
*	y	z

Step 4: If x is 7, y is 6, and z is 3, then y * z is 18. Next, x + 1 is 8. Next, 8 > 18 is False. Next, z == 3 i
True. Finally, False or True is True.

Operator	Left Branch	Right Branch	Mathematical Representation	Evaluation
*	y	z	y * z	18
+	x	1	x + 1	8
>	+	*	8 > 18	False
==	z	3	3 == 3	True
or	>	==	False or True	True

Animation captions:

1. Expressions like x + 1 > y * z or z == 3 are evaluated using precedence rules. Among +, >, *, o
   and ==, the * comes first.
2. Next comes +, >, ==, and finally or.
3. The expression is actually treated like a tree, evaluated from the bottom up.
4. If x is 7, y is 6, and z is 3, then y * z is 18. Next, x + 1 is 8. Next, 8 > 18 is False. Next, z == 3 is
   True. Finally, False or True is True.

PARTICIPATION ACTIVITY 4.10.2: Order of evaluation.

To teach precedence rules, these questions intentionally omit parentheses; good style would use parentheses to make order of evaluation explicit.

1) Which operator is evaluated first?

```
not y and x
```

- `and`
- `not`

2) Which operator has precedence?

```
w + 3 > x - y * z
```

- `+`
- `-`
- `>`
- `*`

3) In what order are the operators evaluated?

```
w + 3 != y - 1 and x
```

- `+`, `!=`, `-`, `and`
- `+`, `-`, `and`, `!=`
- `+`, `-`, `!=`, `and`

4) To what does this expression evaluate, given `x = 4`, `y = 7`.

```
x == 3 or x + 1 > y
```

- True
- False

Common error: Missing parentheses

A common error is to write an expression that is evaluated in a different order than expected. Good practice is to use parentheses in expressions to make the intended order of evaluation explicit. For example, a programmer might write:

- `not a == b` intending to mean `(not a) == b`, but the interpreter computes `not (a == b)` because equality operators (==) have precedence over logical operations (not).

- `w and x == y and z` intending `(w and x) == (y and z)`, but the interpreter computes `(w and (x == y)) and z` because == has precedence over and.
- `not x + y < 5` intending `(not x) + y < 5`, but the interpreter computes `not ((x + y) < 5)` because the addition operator + has the highest precedence and is computed first, followed by the relational operation <, and finally the logical not operation.

PARTICIPATION ACTIVITY 4.10.3: Common errors in expressions.

1) `not x == 3` evaluates as `not (x == 3)`.
   - Yes
   - No

2) `w + x == y + z` evaluates as `(w + x) == (y + z)`.
   - Yes
   - No

3) `w and x == y and z` evaluates as `(w and x) == (y and z)`.
   - Yes
   - No

PARTICIPATION ACTIVITY 4.10.4: Order of evaluation.

Which illustrates the actual order of evaluation via parentheses?

1) `not green == red`
   - `(not green) == red`
   - `not (green == red)`
   - `(not green =)= red`

2) `bats < birds or birds < insects`
   - `((bats < birds) or birds) < insects`
   - `bats < (birds or birds) < insects`

- `(bats < birds) or (birds < insects)`

3) `not (bats < birds) or (birds < insects)`

- `not ((bats < birds) or (birds < insects))`
- `(not (bats < birds)) or (birds < insects)`
- `((not bats) < birds) or (birds < insects)`

4) `(num1 == 9) or (num2 == 0) and (num3 == 0)`

- `(num1 == 9) or ((num2 == 0) and (num3 == 0))`
- `((num1 == 9) or (num2 == 0)) and (num3 == 0)`
- `(num1 == 9) or (num2 == (0 and num3) == 0)`

4.11 Code blocks and indentation

Code blocks

A ***code block*** is a series of statements grouped together. A code block in Python is defined by its indentation level. Ex: the number of blank columns from the left edge. The initial code block is not indented. A new code block can follow a statement that ends with a colon, such as an "if" or "else". In addition, a new code block must be more indented than the previous code block. The program below includes comments indicating where each new code block begins.

The amount of indentation used to indicate a new code block can be arbitrary, as long as the programmer uses the same indentation consistently for each line in the block. *Good practice is to use the standard recommended four columns per indentation level.*

A common error for new Python programmers is the mixing of tabs and spaces. Never mix tabs and spaces for indentation in the same program. Many editors consider a tab to be equivalent to either three or four spaces, while in Python a tab is equivalent only to another tab. A program that mixes tabs and spaces to indent code blocks will automatically generate an IndentationError from the interpreter

in Python 3. *A good practice is to use spaces only when indenting code, and to set text editor options to automatically use spaces when possible.*

Figure 4.11.1: Code blocks are indicated with indentation.

```
# First code block has no indentation

model = input('Enter car model: ')
year = int(input('Enter year of car
manufacture: '))

antique = False
domestic = False

if year < 1970:
    # New code block has indentation of 4
columns
    antique = True

# Back to code block 0

if model in ['Ford', 'Chevrolet', 'Dodge']:
  # New code block has indentation of 2
columns
  # Any amount of indentation > 0 is OK.
  domestic = True

# Back to code block 0

if antique:
    # New code block has indentation of 4
columns
    if domestic:
        # New block has 4 additional
columns (8 total)
        print('My own model-T still runs
like a charm...')
```



```
Enter car model: Ford
Enter year of car manufacture:
1918
My own model-T still runs like
a charm...
```

zyDE 4.11.1: Code blocks and indentation.

Fix the errors in the code below so that you can run the program.

Load default template...

```
1 # Calculate the velocity required to escape a circular or
2 # depending on a user-entered orbital distance.
3
4  import math
```

```
5
6  escape_velocity_meters_per_sec = 0  # Required velocity t
7  grav_constant = 6.67384e-11   # Earth's gravitational cor
8  earth_mass_kilograms = 5.972e24      # Mass of the Earth
9
10     radius_meters = float(input('Enter distance from cent
11     print()
12
13 if radius_meters < 6317000:  # 6317 km is the average rad
14     escape_velocity_meters_per_sec = 0  # No escape possi
15         print('Houston, we have a problem')
16
17
```


160000

Run

Special cases

The acceptable number of columns of text varies from 80 to 120. *Good practice is to use the widely accepted standard of 80 columns.* A few exceptions to the rules of indentation deal with very long statements that require more than one line and *wrap* to the next line. Such special situations do not indicate new code blocks.

Figure 4.11.2: Some indentations are continuations of the previous line.

Multiple lines enclosed within parentheses are implicitly joined into a single string (without newlines between each line); use implicit line joining for very long strings or functions with numerous arguments. Ex: All extra lines are indented to the same column as the opening quotation mark on the first line.

When declaring list or dict literals, entries can be placed on separate lines for clarity.

```
declaration = ("When in the Course of human events, it becomes necessary for
"
               "one people to dissolve the political bands which have
connected "
               "them with another, and to assume among the powers of the
earth...")
```

```
result_of_power = math.pow(long_variable_name_left_operand,
                           long_variable_name_right_operand)
```


Figure 4.11.3: List, dict multi-line constructs.

Containers like lists and dicts can be broken into multiple lines, with the elements on separate, indented lines.

```
my_list = [
    1, 2, 3,
    4, 5, 6
    ]
```

```
my_dict = {
    'entryA': 1,
    'entryB': 2
}
```

PARTICIPATION ACTIVITY 4.11.1: Indentation.

1) The standard number of spaces to use for indentation is four.
   - True
   - False

2) Mixing spaces and tabs when indenting is considered an acceptable programming style.
   - True
   - False

3) A programmer can start new code blocks at any point in the code, as long as the indentation for each line in the block is consistent.
   - True

False

CHALLENGE ACTIVITY 4.11.1: Indentation: Fix the program.

Retype the below code. Fix the indentation as necessary to make the program work.


```
    if 'New York' in temperatures:
        if temperatures['New York'] > 90:
        print('The city is melting!')
        else:
            print(f"The temperature in New York is
{temperatures['New York']}.")
else:
        print('The temperature in New York is unknown.')
```

Ex: If the input is 105, then the output is:

```
The city is melting!
```

Ex: If 'New York' is removed from the dictionary, then the output is:

```
The temperature in New York is unknown.
```

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 temperatures = {
2     'Seattle': 56.5,
3     'New York': float(input()),
4     'Kansas City': 81.9,
5     'Los Angeles': 76.5
6 }
7
8 ''' Your solution goes here '''
9
```


Run

View your last submission

CHALLENGE ACTIVITY | 4.11.2: Code blocks and indentation.

566436.4601014.qx3zqy7

Start

The following code contains at least one indentation error. Find and fix the error(s) so the program pr and perimeter of a rectangle.

▶ **Click here for example**

```
length = int(input())
width = int(input())
area = length * width
perimeter = 2 * (length + width)

# Fix the indentation in the code below
    print(f'Length = {length} in')
                print(f'Width = {width} in')
    print(f'Area = {area:.3f} in^2')
            print(f'Perimeter = {perimeter:.3f} in')
```

1 2 3

Check Next level

4.12 Conditional expressions

Conditional expressions

A ***conditional expression*** has the following form:

Construct 4.12.1: Conditional expression.

```
expr_when_true if condition else
expr_when_false
```


All three operands are expressions. The condition in the middle is evaluated first. If the condition evaluates to True, then expr_when_true is evaluated. If the condition evaluates to False, then expr_when_false is evaluated. For example, if x is 2, then the conditional expression `5 if x==2 else 9*x` evaluates to 5.

A conditional expression has three operands and thus is sometimes referred to as a ***ternary operation***.

Good practice is to restrict usage of conditional expressions to an assignment statement, as in: `y = 5 if (x == 2) else 9*x`. Some Python programmers denounce conditional expressions as difficult to read and comprehend, since the middle operand is actually the first evaluated, and left-to-right syntax is preferred. However, simple assignments such as the statement above are acceptable.

PARTICIPATION ACTIVITY 4.12.1: Conditional expression.

```
if condition:
   my_var = expr1
else:
   my_var = expr2
```

```
my_var = expr1 if (condition) else  expr2
```


Animation content:

Static figure:
Begin Python code:
if condition:
   my_var = expr1
else:
   my_var = expr2

End Python code.

Begin Python code:
my_var = expr1 if (condition) else expr2
End Python code.

Animation captions:

1. This if-else form can be written as a conditional expression.
2. The condition in the middle is evaluated first.
3. If the condition evaluates to True, then expr1 is evaluated and my_var is assigned with expr1's value.
4. If the condition evaluates to False, then expr2 is evaluated and my_var is assigned with expr2's value.

PARTICIPATION ACTIVITY 4.12.2: Conditional expressions.

Convert each if-else statement to a single assignment statement using a conditional expression with parentheses around the condition. Enter "Not possible" if appropriate.

1)
```
if x < 100:
    y = 0
else:
    y = x
```

Check Show answer

2)
```
if x < 0:
    x = -x
else:
    x = x
```

Check Show answer

3)
```
if x < 1:
    y = x
else:
    z = x
```

Check Show answer

CHALLENGE ACTIVITY 4.12.1: Conditional expressions: Enter the output of the code.

566436.4601014.qx3zqy7

Start

Type the program's output

```
my_number = 1
your_number = 3 if my_number > 5 else 9
print(your_number)
```

9

1 2

Check Next

CHALLENGE ACTIVITY 4.12.2: Conditional expression: Print negative or nonnegative.

Create a conditional expression that evaluates to string "negative" if user_val is less than 0, and "nonnegative" otherwise.

Sample output with input: -9

```
-9 is negative
```

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 user_val = int(input())
2
3 cond_str = ''' Your solution goes here '''
4
5 print(f'{user_val} is {cond_str}')
```

Run

View your last submission

CHALLENGE ACTIVITY 4.12.3: Conditional expression: Conditional assignment.

Using a conditional expression, write a statement that increments num_users if update_direction is 3, otherwise decrements num_users.

Sample output with inputs: 8 3

```
New value is: 9
```

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 num_users = int(input())
2 update_direction = int(input())
3
4 num_users = ''' Your solution goes here '''
5
6 print(f'New value is: {num_users}')
```


Run

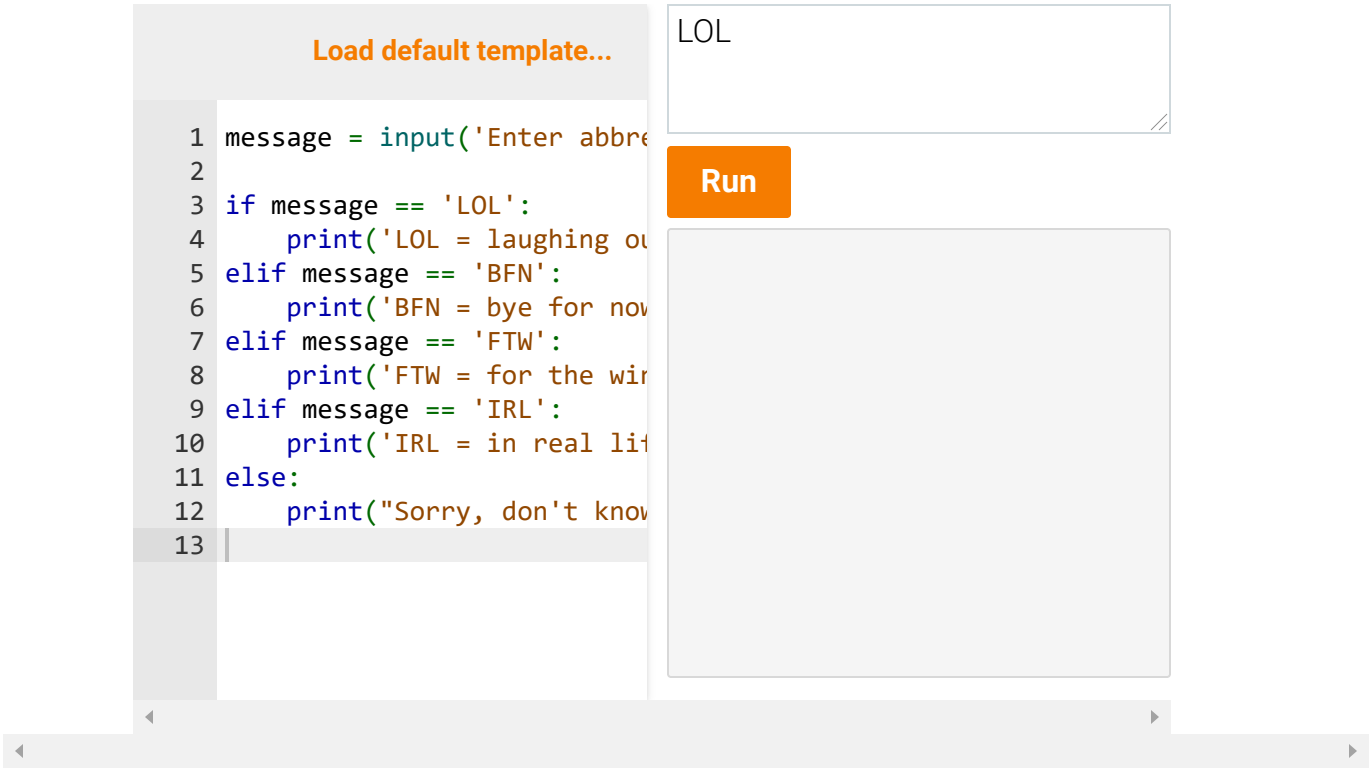
View your last submission

4.13 Additional practice: Tweet decoder

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following program decodes a few common abbreviations in online communication such as messages in Twitter ("tweets") or email, and provides the corresponding English phrase.

zyDE 4.13.1: Abbreviation decoder.



Create different versions of the program that:

1. Expand the number of abbreviations that can be decoded. Add support for abbreviations you commonly use.
2. Allow the user to enter a complete tweet (160 characters or less) as a single line of text. Search the resulting string for abbreviations and print a list of each abbreviation along with its decoded meaning.

4.14 LAB: Smallest number

This section's content is not available for print.

4.15 LAB: Interstate highway numbers

Primary U.S. interstate highways are numbered 1-99. Odd numbers (like the 5 or 95) go north/south, and evens (like the 10 or 90) go east/west. Auxiliary highways are numbered 100-999, and service the primary highway indicated by the rightmost two digits. Thus, I-405 services I-5, and I-290 services I-90. Note: 200 is not a valid auxiliary highway because 00 is not a valid primary highway number.

Given a highway number, indicate whether it is a primary or auxiliary highway. If auxiliary, indicate what primary highway it serves. Also indicate if the (primary) highway runs north/south or east/west.

Ex: If the input is:

```
90
```

the output is:

```
I-90 is primary, going east/west.
```

Ex: If the input is:

```
290
```

the output is:

```
I-290 is auxiliary, serving I-90, going east/west.
```

Ex: If the input is:

```
0
```

the output is:


```
0 is not a valid interstate highway number.
```

Ex: If the input is:

```
200
```

the output is:

```
200 is not a valid interstate highway number.
```

See Wikipedia for more info on highway numbering.

566436.4601014.qx3zqy7

LAB ACTIVITY | 4.15.1: LAB: Interstate highway numbers 10 / 10

main.py

```
1 Loading latest submission...
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

4.16 LAB: Seasons

This section's content is not available for print.

4.17 LAB: Exact change

This section's content is not available for print.

4.18 LAB: Leap year

This section's content is not available for print.

4.19 LAB: Golf scores

Golf scores record the number of strokes used to get the ball in the hole. The expected number of strokes varies from hole to hole and is called par (possible values: 3, 4, or 5). Each score's name is based on the actual strokes taken compared to par:

- "Eagle": number of strokes is two less than par
- "Birdie": number of strokes is one less than par
- "Par": number of strokes equals par
- "Bogey": number of strokes is one more than par

Given two integers that represent the number of strokes used and par, write a program that prints the appropriate score name. Print "Error" at the end of the output if par is not 3, 4, or 5, or if the score's name is not "Eagle", "Birdie", "Par", or "Bogey".

Ex: If the input is:

```
3
4
```

the output is:

```
Par 4 in 3 strokes is Birdie
```


Ex: If the input is:

```
2
1
```

the output is:

```
Par 1 in 2 strokes is Error
```


LAB ACTIVITY 4.19.1: LAB: Golf scores 10 / 10

main.py

```
1 Loading latest submission...
```

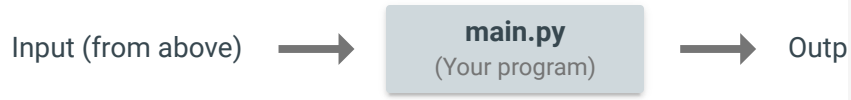

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Coding trail of your work What is this?

Retrieving signature