4.1 If-else branches (general)

Branch concept

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

PARTICIPATION ACTIVITY 4.1.1: Branching concept.

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. Other size parties are seated at a large table.
3. The host mentally executes the algorithm: If party of 1, seat at counter; Else If party of 2, seat at small table; Else seat at 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)

A ***branch*** is a program path taken only if an expression's value is true. Ex: A hotel may discount a price only for people over age 60.

PARTICIPATION ACTIVITY 4.1.3: A simple branch: Hotel discount.

Animation captions:

1. A branch is a program path taken if an expression is true. This program assigns price = 155, and gets input age. If age > 60, the branch with price = price - 20 should be taken.
2. When executing, if the input age is 72, then age > 60 is true. The branch is taken, so price = price - 20 executes. The output is thus 135.
3. But, if the input age were 45, then age > 60 is false. The branch is not taken, so the output is 155.

Such a branch is commonly known as an ***If*** branch: A branch taken only if an expression is true.

PARTICIPATION ACTIVITY 4.1.4: Branches.

Consider the example above.

1) If the input is 25, the output is Cost: ____ .
   - ○ 135
   - ○ 155

2) If the input is 80, the output is Cost: ____ .
   - ○ 135
   - ○ 155

3) If the input is 60, will the branch be taken?
   - ○ Yes
   - ○ No

If branch example: Absolute value

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

PARTICIPATION ACTIVITY 4.1.5: Example if branch: Computing absolute value.

Animation captions:

1. This program outputs the absolute value of the input. After getting input val, if val < 0, the branch val = -val executes. Finally, val is output.
2. When executing, if the input val is -9, then val < 0 is true. The branch is taken, so val = -val executes, assigning val with 9. So 9 is output.
3. But, if the input val were 45, then val < 0 is false. The branch is not taken, so the output is 45.

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*** structure has two branches: The first branch is taken if an expression is true, else the other branch is taken.

PARTICIPATION ACTIVITY 4.1.7: If-else branches.

Animation captions:

1. An if-else has two branches. The if branch is taken if the expression is true. Otherwise, the else branch is taken.
2. This program should output "Can ride. Done." if the input weight < 150, else should output "Can't ride. Done."
3. If the input weight is 128, weight < 150 is true, so the if branch executes. "Can ride" is output.
4. But, if the input is 175, weight < 150 is false, so the else branch executes. "Can't ride" is output.

PARTICIPATION ACTIVITY 4.1.8: If-else branches.

Consider the example above.

1) What is output when the input is 90?
   - ○ Can ride. Done.
   - ○ Can't ride. Done.

2) What is output when the input is 200?
   - ○ Can ride. Done.
   - ○ Can't ride. Done.

3) What is output when the input is exactly 150?
   - ○ Can ride. Done. Can't ride. Done.
   - ○ Can ride. Can't ride. Done.
   - ○ Can't ride. Done.

4) What input value causes both the if branch to execute (outputting "Can ride") and the else branch to execute (outputting "Can't ride")?
   - ○ 149
   - ○ 150
   - ○ 151
   - ○ No such value

5) What value causes "Done." to NOT be output?
   - ○ 130
   - ○ 160
   - ○ No such value.

If-else example: Max

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

PARTICIPATION ACTIVITY 4.1.9: If-else example: Max.

Animation captions:

1. This program should output the maximum of two input values.
2. If x is 7 and y is 3, x > y is true, so the if branch executes, which assigns max with x, so 7.
3. If the input instead is 3 7, x > y is false, so the else executes. max is assigned with y, which is 7. Thus, whether the input is 7 3 or is 3 7, max gets the larger value.

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.

PARTICIPATION ACTIVITY 4.1.11: If-elseif-else branch.

Animation captions:

1. This program gets a year, then if 1984 outputs "Orwell!", else if 2001 outputs "Space Odyssey!", else outputs "Nothing special", thus indicating if a year was featured in a famous book/movie.
2. If year equals 1984, the if branch is taken, outputting "Orwell!".
3. If the input is 2001, year equals 1984 is false. So the next branch is checked: year equals 2001 is true, so that branch is taken, thus outputting "Space Odyssey!".
4. If year equals anything else, like 1997, the if branch is not taken, and else-if branch is not taken, so the else branch is taken. "Nothing special" is output.

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

4.2 If-else statement

If statements

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

PARTICIPATION ACTIVITY 4.2.1: If statement: Hotel discount.

Animation captions:

1. An if statement executes a group of statements if an expression is true. The program assigns hotel_rate with 155 and then gets the user's age from input.
2. user_age is 68, so the expression 68 > 65 is true, and the if's statement will execute. Thus, the statement following the colon : will execute next.
3. hotel_rate is decreased by 20, which is the discount for guests older than 65.

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 = 19
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

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.

Construct 4.2.1: If-else statement.

```
# Statements that execute before the branches

if expression:
   # Statements that execute when expression is true (first branch)
else:
   # Statements that execute when expression is false (second branch)

# Statements that execute after the branches
```

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

PARTICIPATION ACTIVITY 4.2.3: If-else statement: Car insurance.

Animation content:

undefined

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_age is 22, so the expression 22 < 25 is true, and the if's statements will execute.
3. user's age is 40, so the expression 40 < 25 is false, and the else's statements will execute.

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, 2009l)

PARTICIPATION ACTIVITY 4.2.4: If-else statements.

1) What is the final value of num_items?

```
bonus_val = 5

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 = 12

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 < 12:
   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 + 2
else:
   bonus_val = bonus_val + 10
```

Check Show answer

5) What is the final value of bonus_val?

```
bonus_val = 11

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 4.2.5: 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 some consistent number of spaces, such as 3 spaces.)

1) If user_age is greater than 62, assign item_discount with 15. Else, assign item_discount with 0.

Check **Show answer**

2) If num_people is greater than 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 is greater than 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**

©zyBooks 03/05/20 10:24 591419
Alexey Munishkin
UCSCCSE20NawabWinter2020

CHALLENGE ACTIVITY 4.2.1: Enter the output for the if-else branches.

CHALLENGE ACTIVITY 4.2.2: Basic if-else expression.

Write an expression that will cause the following code to print "18 or less" if the value of user_age is 18 or less. Write *only* the expression.

Sample output with input: 17

```
18 or less
```



```
user_age = int(input())
if ''' Your solution goes here ''':
   print('18 or less')
else:
   print('Over 18')
```

Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 4.2.3: Basic if-else.

Write an if-else statement for the following:
If user_tickets is less than 5, assign 1 to num_tickets. Else, assign user_tickets to num_tickets.

Sample output with input: 3

```
Value of num_tickets: 1
```

```
user_tickets = int(input())

''' Your solution goes here '''

print('Value of num_tickets:', num_tickets)
```


Run

View solution *(Instructors only)*

Download student submissions

Multi-branch if-else statements

An if-else statement can be extended to have three (or more) branches. 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.

Construct 4.2.2: 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)
```

The ***equality operator ==*** evaluates to true if the left side and right side are equal. Ex: If num_years holds the value 10, then the expression `num_years == 10` evaluates to true.

Note that the equality operator is ==, not =.

Note: Good Practice is to be as explicit as possible and reduce the chances of logical errors by using the less-than-or-equal-to rather than just less-than. Thus, the constant is the number of relevance, e.g., less-than-or-equal-to 39 rather than less-than 40, because 40 is not the relevant number.

Figure 4.2.1: 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 4.2.6: 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 4.2.4: Multi-branch if-else statements: Print century.

Write an if-else statement with multiple branches.
If year is 2101 or later, print "Distant future" (without quotes). Otherwise, if year is 2001 or greater, print "21st century". Otherwise, if year is 1901 or greater, print "20th century". Else (1900 or earlier), print "Long ago".

Sample output with input: 1776

```
Long ago
```

```
1 year = int(input())
2
3 ''' Your solution goes here '''
4
```


Run

View solution *(Instructors only)*

Download student submissions

4.3 More if-else

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.

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

PARTICIPATION ACTIVITY 4.3.1: Nested if-else

PARTICIPATION ACTIVITY 4.3.2: 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

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 thus more than one branch can execute, in contrast to the multi-branch if-else arrangement.

PARTICIPATION ACTIVITY 4.3.3: Multiple distinct if statements.

PARTICIPATION ACTIVITY 4.3.4: 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 = 9

if num_apples < 10:
   if num_apples < 5:
      num_boxes = 1
   else:
      num_boxes = 2
elif num_apples < 20:
   num_boxes = num_boxes + 1
```

Check Show answer

CHALLENGE ACTIVITY 4.3.1: Enter the output for the multiple if-else branches.

CHALLENGE ACTIVITY 4.3.2: Multiple if statements: Printing car features.

Write multiple if statements. If car_year is 1969 or earlier, print "Few safety features." If 1970 or later, print "Probably has seat belts." If 1990 or later, print "Probably has antilock brakes." If 2000 or later, print "Probably has airbags." End each phrase with a period and a newline.

Sample output for input: 1995

```
Probably has seat belts.
Probably has antilock brakes.
```

```
1 car_year = int(input())
2
3 ''' Your solution goes here '''
4
```


Run

View solution *(Instructors only)*

Download student submissions

4.4 Equality and relational operators

Equality operators

An ***equality operator*** checks whether two operands' values are the same (==) or different (!=). Note that equality is ==, not just =.

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

Table 4.4.1: Equality 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 4.4.1: 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 4.4.2: Creating expressions with equality operators.

Type the operator to complete the desired expression.

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

Relational operators

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

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

Table 4.4.2: 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 4.4.3: 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 4.4.4: 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**

zyDE 4.4.1: If-else with expression: Non-negative.

The program prints "Zero" if the user enters 0, else prints "Non-zero". Modify the program print "Non-negative" if the user enters 0 or greater, else print "Negative".

Load default template...

```
99
```

Run

CHALLENGE ACTIVITY 4.4.1: Enter the output for the branches with relational and equality operators.

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

if user_num == 0:
    print('Zero')
else:
    print('Non-zero')

```

CHALLENGE ACTIVITY 4.4.2: Relational operators.

Write an expression that will print "Dollar or more" if the value of num_cents is at least a dollar (100 cents is a dollar).

Sample output with input: 109

```
Dollar or more
```

```
num_cents = int(input())
if ''' Your solution goes here ''':
    print('Dollar or more')
else:
    print('not a dollar')
```

Run

View solution *(Instructors only)*

Download student submissions

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 4.4.5: 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 non-negative number less than 100.

```
if :
    # evaluated to true
else:
    # evaluated to false
```

Check Show answer

CHALLENGE ACTIVITY 4.4.3: If-else expression: Operator chaining.

Write an expression that will print "in high school" if the value of user_grade is between 9 and 12 (inclusive).

Sample output with input: 10

```
in high school
```

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

Run

View solution *(Instructors only)*

Download student submissions

Comparing characters, strings, and floating-point types

The relational and equality operators work for integer, character, 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, as discussed in a later section.

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 4.4.6: Comparing various types.

Which comparison 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_double == 3.25
   - 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, like `5 < 2`, or `'abc' >= 'ABCDEF'`, depend 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 order is determined by the first mismatching elements in the sequences. For example, if x = [1,2] and y = [1,3], then evaluating `x < y` first evaluates `1 < 1`, then `2 < 3`, which produces a value of True.
- Dictionaries are compared by sorting the keys and values of each dictionary and then comparing them as lists.

PARTICIPATION ACTIVITY 4.4.7: 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 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 4.4.8: 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

4.5 Boolean operators and expressions

Booleans and Boolean 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 the boolean value True to my_bool

my_val  = 5
is_small = my_val < 3 # Evaluates the expression and assigns False to is_small
```


A ***Boolean operator*** treats operands as True or False and evaluates to a value of True or False. Boolean operators include `and`, `or`, and `not`. A ***Boolean expression*** is an expression using Boolean operators.

PARTICIPATION ACTIVITY 4.5.1: Boolean operators: and, or, and not.

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 commonly an expression itself. If x = 7, y = 9, then (x > 0) and (y < 10) is True and True, so evaluates to true (both operands are True).

Table 4.5.1: Boolean operators.

Boolean 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 4.5.2: Boolean 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 4.5.2: Evaluating expressions with Boolean operators.

Given num_people = 10, num_cars = 2, user_key = 'q'.

1) `num_people >= 10`

- ○ True
- ○ False

2) `(num_people >= 10) and (num_cars > 2)`

- ○ True
- ○ False

3) `(num_people >= 20) or (num_cars > 1)`

True

False

4) `not (num_cars < 5)`

True

False

5) `not (user_key == 'a')`

True

False

6) `user_key != 'a'`

True

False

7) `not ((num_people >= 10) and (num_cars > 2))`

True

False

8) `(user_key == 'x') or ((num_people > 5) and (num_cars > 1))`

True

False

Boolean operators are commonly used in expressions found in if-else statements.

PARTICIPATION ACTIVITY 4.5.3: Boolean operators: Complete the expressions for the given condition.

Fill in the missing Boolean operators. Assume all variables are integers.

1) days is greater than 30 and less than 90

```
if (days > 30) [ ] (days <
90):
  ...
```

Check Show answer

2) max_cars is between 0 and 100.

```
if (max_cars > 0) [ ]
(max_cars < 100):
  ...
```

Check Show answer

3) num_stores is between 10 and 20, inclusive.

```
if (num_stores >= 10) and (
[                        ]):
  ...
```

Check Show answer

4) num_dogs is 3 or more and num_cats is 3 or more.

```
if (num_dogs >= 3)
                  :
  ...
```

Check Show answer

5) Either wage is greater than 10 or age is less than 18. Use "or". Use > and < (not >= and <=). Use parentheses around sub-expressions.

```
if                  :
  ...
```

Check Show answer

6) num is a 3-digit positive integer, such as 100, 989, or 523, but not 55, 1000, or -4.

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

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

Check Show answer

CHALLENGE ACTIVITY 4.5.1: Boolean operators: Detect specific values.

Write an expression using Boolean operators that prints "Special number" if special_num is -99, 0, or 44.

Sample output with input: 17

```
Not special number
```

```
special_num = int(input())

if ''' Your solution goes here ''':
    print('Special number')
else:
    print('Not special number')
```


Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 4.5.2: Boolean operators: Combining test conditions.

Write an expression using Boolean operators that prints "Eligible" if user_age is greater than 17 and not equal to 25.

Sample output with input: 17

```
Ineligible
```

```
1 user_age = int(input())
2
3 if ''' Your solution goes here ''':
4     print('Eligible')
5 else:
6     print('Ineligible')
```

Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 4.5.3: Boolean operators: Branching using Boolean variables.

Write an expression that prints 'You must be rich!' if the variables young and famous are both True.

Sample output with inputs: 'True' 'True'

```
You must be rich!
```

Run

View solution *(Instructors only)*

Download student submissions

4.6 Membership and identity operators

Membership operators: in/not in

One common 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 some element in the right operand, which is always a container.

PARTICIPATION ACTIVITY 4.6.1: Membership operators: Checking for a value in a list.

Animation captions:

1. A user creates a new list.
2. Every element in the list is checked for the value of 14.99.

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 i for which `a == x` is True. The program below demonstrates membership operator usage in a list:

Figure 4.6.1: 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('Found', name, 'on the roster.')
else:
    print('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('Could not find', name, 'on the
roster.')
else:
    print('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 4.6.2: 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 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 4.6.3: 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 4.6.2: 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 [1] 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 and thus 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 4.6.4: Identity operators.


```
x = 500 + 500  # Create a new object with value 1000
y = 500 + 500  # 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('but z and y are not.')
```

```
z and x are bound to the same object,
but z and y are not.
```

PARTICIPATION ACTIVITY 4.6.3: 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 4.6.1: 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, or 44].

Sample output with input: 17

```
Not special number
```

```
special_list = [-99, 0, 44]
special_num = int(input())

if ''' Your solution goes here ''':
    print('Special number')
else:
    print('Not special number')
```


Run

View solution *(Instructors only)*

Download student submissions

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

4.7 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 4.7.1: Precedence rules for arithmetic, logical, and relational operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In `(a * (b + c)) - d`, the + is evaluated first, then *, then -.
* / % + -	Arithmetic operators (using their precedence rules; see earlier section)	`z - 45 * y < 53` evaluates * first, then -, then <.
`< <= > >= == !=`	Relational, (in)equality, and	`x < 2 or x >= 10` is evaluated as `(x < 2) or (x >= 10)` because < and >= have precedence over or.

	membership operators	
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 4.7.1: Applying the precedence rules to an expression can be thought of as a 'tree'.

Animation captions:

1. Expressions like x + 1 > y * z or z == 3 are evaluated using precedence rules. Among +, >, *, or, and ==, the * comes first.
2. Next comes +, then ==, and finally or.
3. The expression is actually treated like a "tree", evaluated from the bottom upwards.
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 4.7.2: 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 is evaluated first?
`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 in fact 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 4.7.3: 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 4.7.4: 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)`

4.8 Code blocks and indentation

Code blocks

A ***code block*** is a series of statements that are grouped together. A code block in Python is defined by its indentation level, i.e., 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 4 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 3 or 4 spaces, while in Python a tab is equivalent only to another tab. A program that mixes tabs and space 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.

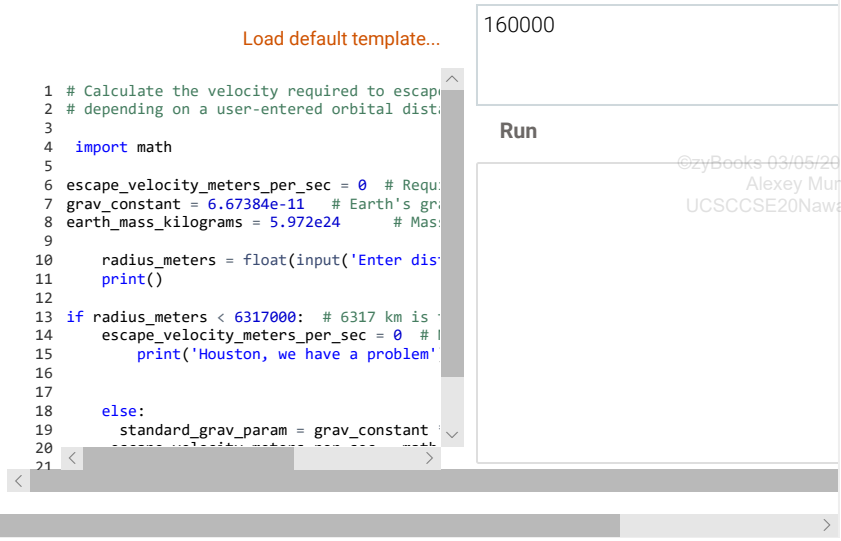
Figure 4.8.1: Code blocks are indicated with indentation.

```
# 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 4.8.1: Code blocks and indentation.

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



Special cases

The number of columns of text considered to be acceptable 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 4.8.2: 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 4.8.3: 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 4.8.1: Indentation.

1) The standard number of spaces to use for indentation is 4.

- ○ 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 4.8.1: 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('The temperature in New York is', temperatures['New
York'])
else:
        print('The temperature in New York is unknown.')
```

Sample output with input: 105

```
The city is melting!
```

```
temperatures = {
    'Seattle': 56.5,
    'New York': float(input()),
    'Kansas City': 81.9,
    'Los Angeles': 76.5
}

''' Your solution goes here '''

```


Run

View solution *(Instructors only)*

Download student submissions

4.9 Conditional expressions

Conditional expressions

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

Construct 4.9.1: 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 4.9.1: Conditional expression.

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 its value is assigned to my_var.
4. If the condition evaluates to False, then expr2 is evaluated and its value is assigned to my_var.

PARTICIPATION ACTIVITY 4.9.2: Conditional expressions.

Convert each if-else statement to a single assignment statement using a conditional expression, using 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 4.9.1: Conditional expression: Print negative or non-negative.

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

Sample output with input: -9

```
-9 is negative
```

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

Run

View solution *(Instructors only)*

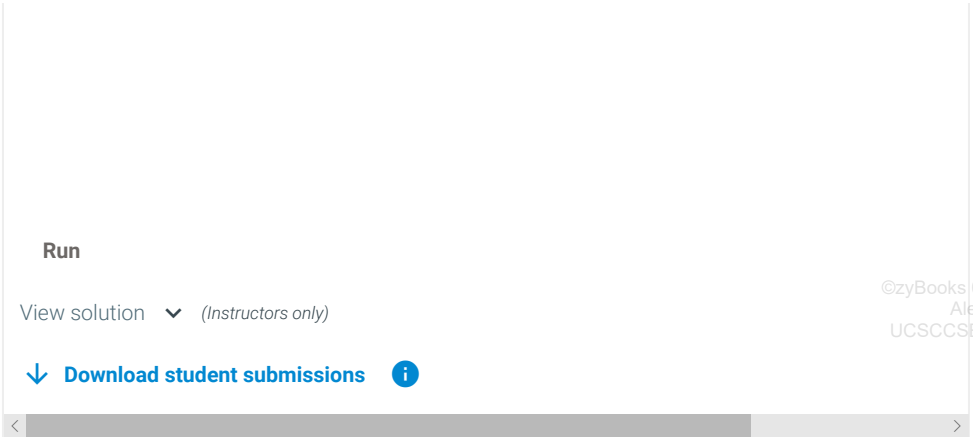
Download student submissions

CHALLENGE ACTIVITY 4.9.2: 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
```



4.10 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 4.10.1: Tweet decoder.

Load default template...

```
1  tweet = input('Enter abbreviation from tweet:
2
3  if tweet == 'LOL':
4      print('LOL = laughing out loud')
5  elif tweet == 'BFN':
6      print('BFN = bye for now')
7  elif tweet == 'FTW':
8      print('FTW = for the win')
9  elif tweet == 'IRL':
10     print('IRL = in real life')
11 else:
12     print("Sorry, don't know that one")
13
```

LOL

Run

Create different versions of the program that:

1. Expand the number of abbreviations that can be decoded. Add support for abbreviations you commonly use or search the internet for some.
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.

4.11 LAB: Remove gray from RGB

Summary: Given integer values for red, green, and blue, subtract the gray from each value.

Computers represent color by combining the sub-colors red, green, and blue (rgb). Each sub-color's value can range from 0 to 255. Thus (255, 0, 0) is bright red, (130, 0, 130) is a medium purple, (0, 0, 0) is black, (255, 255, 255) is white, and (40, 40, 40) is a dark gray. (130, 50, 130) is a faded purple, due to the (50, 50, 50) gray part. (In other words, equal amounts of red, green, blue yield gray).

Given values for red, green, and blue, remove the gray part.

Ex: If the input is:

```
130
50
130
```


the output is:

```
80 0 80
```

Find the smallest value, and then subtract it from all three values, thus removing the gray.

Note: This page converts rgb values into colors.

LAB ACTIVITY 4.11.1: LAB: Remove gray from RGB 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.12 LAB: Smallest number

Write a program whose inputs are three integers, and whose output is the smallest of the three values.

Ex: If the input is:

```
7
15
3
```

the output is:

```
3
```

LAB ACTIVITY 4.12.1: LAB: Smallest number 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.13 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.

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.
```

See Wikipedia for more info on highway numbering.

LAB ACTIVITY 4.13.1: LAB: Interstate highway numbers 0 / 10

main.py Load default template...

```
highway_number = int(input())

''' Type your code here. '''

```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

4.14 LAB: Seasons

Write a program that takes a date as input and outputs the date's season. The input is a string to represent the month and an int to represent the day.

Ex: If the input is:

```
April
11
```

the output is:


```
Spring
```

In addition, check if the string and int are valid (an actual month and day).

Ex: If the input is:

```
Blue
65
```

the output is:

```
Invalid
```

The dates for each season are:
Spring: March 20 - June 20
Summer: June 21 - September 21
Autumn: September 22 - December 20
Winter: December 21 - March 19

LAB ACTIVITY 4.14.1: LAB: Seasons 0 / 10

main.py Load default template...

```
1 input_month = input()
2 input_day = int(input())
3
4 ''' Type your code here. '''
5
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

4.15 LAB: Exact change

Write a program with total change amount as an integer input, and output the change using the fewest coins, one coin type per line. The coin types are Dollars, Quarters, Dimes, Nickels, and Pennies. Use singular and plural coin names as appropriate, like 1 Penny vs. 2 Pennies.

Ex: If the input is:

```
0
```


(or less than 0), the output is:

```
No change
```

Ex: If the input is:

```
45
```

the output is:

```
1 Quarter
2 Dimes
```

LAB ACTIVITY 4.15.1: LAB: Exact change 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
2
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.16 LAB: Leap year

A year in the modern Gregorian Calendar consists of 365 days. In reality, the earth takes longer to rotate around the sun. To account for the difference in time, every 4 years, a leap year takes place. A leap year is when a year has 366 days: An extra day, February 29th. The requirements for a given year to be a leap year are:

1) The year must be divisible by 4

2) If the year is a century year (1700, 1800, etc.), the year must be evenly divisible by 400

Some example leap years are 1600, 1712, and 2016.

Write a program that takes in a year and determines whether that year is a leap year.

Ex: If the input is:

```
1712
```

the output is:

```
1712 - leap year
```

Ex: If the input is:

```
1913
```

the output is:

```
1913 - not a leap year
```

LAB ACTIVITY 4.16.1: LAB: Leap year 0 / 10

main.py Load default template...

```
1 is_leap_year = False
2
3 input_year = int(input())
4
5 ''' Type your code here. '''
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.17 LAB: Warm up: Automobile service cost

(1) Prompt the user for an automobile service. Output the user's input. (1 pt)

Ex:

```
Enter desired auto service:
Oil change
You entered: Oil change
```

(2) Output the price of the requested service. (4 pts)

Ex:

```
Enter desired auto service:
Oil change
You entered: Oil change
Cost of oil change: $35
```


The program should support the following services (all integers):

- Oil change -- $35
- Tire rotation -- $19
- Car wash -- $7

If the user enters a service that is not listed above, then output the following error message:

```
Error: Requested service is not recognized
```

LAB ACTIVITY 4.17.1: LAB: Warm up: Automobile service cost 0 / 5

main.py Load default template...

```
1 # Type your code here
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.18 LAB*: Program: Automobile service invoice

(1) Output a menu of automotive services and the corresponding cost of each service. (2 pts)

Ex:

```
Davy's auto shop services
Oil change -- $35
Tire rotation -- $19
Car wash -- $7
Car wax -- $12
```

(2) Prompt the user for two services from the menu. (2 pts)

Ex:

```
Select first service:
Oil change
Select second service:
Car wax
```

(3) Output an invoice for the services selected. Output the cost for each service and the total cost. (3 pts)

```
Davy's auto shop invoice

Service 1: Oil change, $35
Service 2: Car wax, $12

Total: $47
```

(4) Extend the program to allow the user to enter a dash (-), which indicates no service. (3 pts)

Ex:

```
Davy's auto shop services
Oil change -- $35
Tire rotation -- $19
Car wash -- $7
Car wax -- $12

Select first service:
Tire rotation
Select second service:
-

Davy's auto shop invoice

Service 1: Tire rotation, $19
Service 2: No service

Total: $19
```


LAB ACTIVITY 4.18.1: LAB*: Program: Automobile service invoice 0 / 10

main.py

```
1 # Type your code here
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.19 Programming Assignment 2 - Problem 1

Problem 1 - FIZZ BUZZ (30 points)

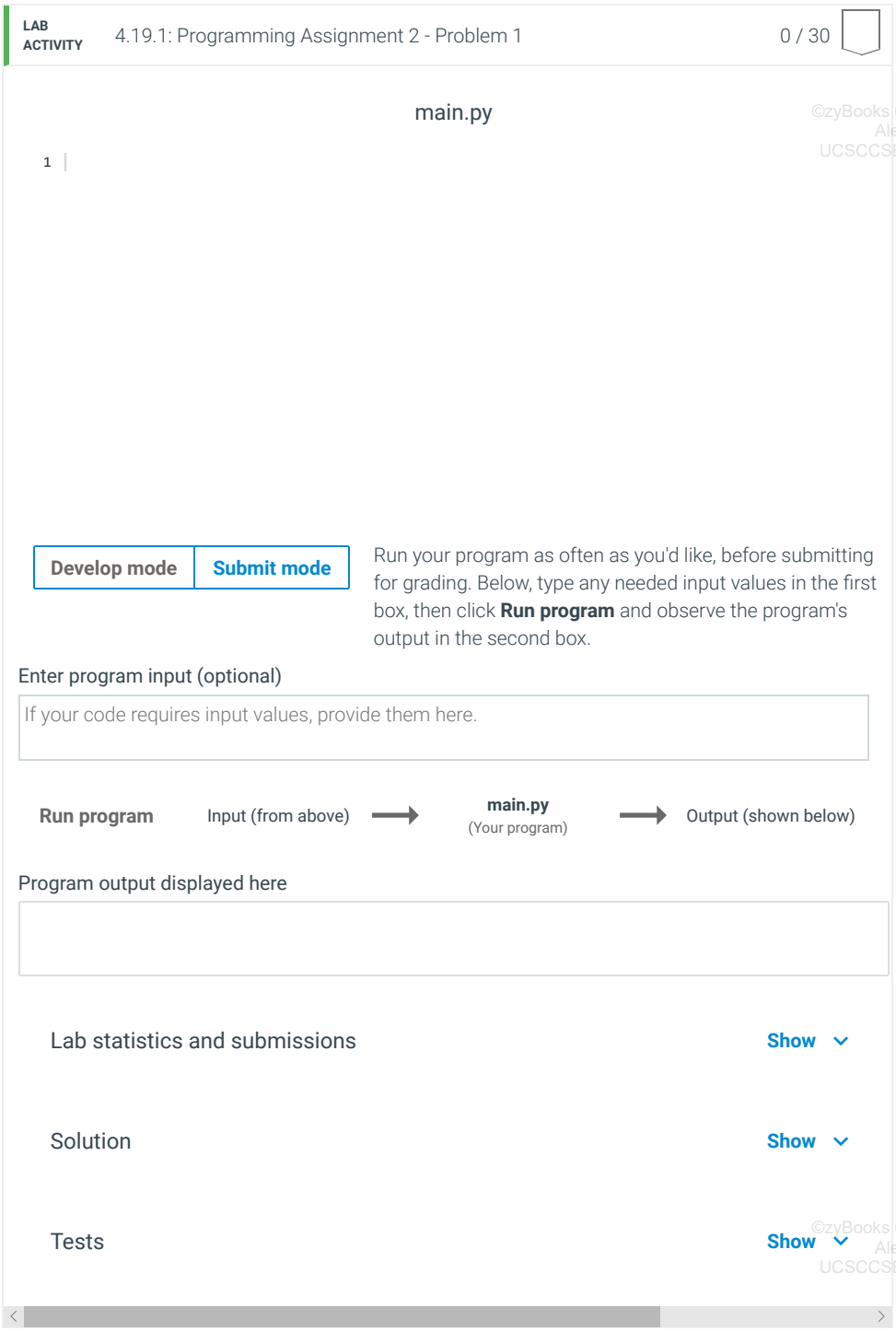
Write a script that reads in two integers from the user, then prints all the numbers from 1 to 20, with the following exceptions: If a number is divisible by ....

```
     (a) the first entered integer, print "FIZZ" instead of the number.
     (b) the second entered integer, print "BUZZ" instead of the number.
     (c) both the first AND the second integers, print "FIZZBUZZ" instead of the
number.
```

For example, if the inputs are 3 and 5 , the output should be:

1 2 FIZZ 4 BUZZ FIZZ 7 8 FIZZ BUZZ 11 FIZZ 13 14 FIZZBUZZ 16 17 FIZZ 19 BUZZ

(Note that each item is separated by a single space character and that the expected output has a trailing space followed by a newline character.)



4.20 Programming Assignment 2 - Problem 2

Problem 2 - Passphrase Check (40 points)

We are trying to make sure your passphrase is acceptable! Write a script that asks for an input string that will be used as your password. This input string will have three "words" separated by a space character. Each of these words has specific requirements, listed below. You must write a function for each requirement (three total functions). Make sure your functions follow the original template that provided.

First Function:

For the first word, you must check if it is a palindrome (it is the same forward and backward, like "glolg" or "tacocat"), is case-insensitive (so "GfraaRFg" or "VObov" are valid), and only uses alphabetical letters ("tre3ert" and "f^h,@s@,h^f" are not valid).

Second Function:

For the second word, you must check if it is a positive integer with at least 4 digits, containing only numeric characters (i.e. no alphabetic, punctuation, or special characters).

Third Function:

For the third word, you must check if it starts with one of these special characters ("@", "$", "%", "^", "&"), ends with a punctuation mark (only commas (","), periods ("."), exclamation marks ("!") and question marks ("?") are allowed), and has three letters in between the special character and punctuation mark (e.g. "@dog!" or "$tpb?" or "^cat,").

If the entire password meets all the correct specifications, then print "valid"; otherwise, print "invalid".

Template Functions (DO NOT CHANGE FUNCTION NAMES):

```
def palindrome():
def integer():
def special():
```

LAB ACTIVITY 4.20.1: Programming Assignment 2 - Problem 2 0 / 40

main.py

```
1
```

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.

zyBooks 03/05/20 10:24 591419
Alexey Munishkin
UCSCCSE20NawabWinter2020

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.py** (Your program) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

4.21 Programming Assignment 2 - Problem 3

Problem 3 - Writing a Multiples List (30 points)

Write a program that takes in two positive integer numbers (N) and (D) and prints out the first N numbers divisible by D. The final output should be a list of N numbers divisible by D. For example, if you count up from 1, and are given N = 3 and D = 4, then your output list should be [4, 8, 12].

LAB ACTIVITY 4.21.1: Programming Assignment 2 - Problem 3 0 / 30

main.py

```
1
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

©zyBooks 03/05/20 10:24 591419
Alexey Munishkin
UCSCCSE20NawabWinter2020