3.1 String basics

Strings and string literals

A ***string*** is a sequence of characters, like the text MARY, that can be stored in a variable. A ***string literal*** is a string value specified in the source code of a program. A programmer creates a string literal by surrounding text with single or double quotes, such as 'MARY' or "MARY".

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

The string type is a special construct known as a ***sequence type***: A type that specifies a collection of objects ordered from left to right. A string's characters are ordered from the string's first letter to the last. A character's position in a string is called the character's index, which starts at 0. Ex: In "Trish", T is at index 0, r at 1, etc.

PARTICIPATION ACTIVITY 3.1.1: String indexing.

Type a string below to see how a string is a sequence of characters ordered by position. The numbers on top indicate each character's index.

Type a string
(up to 6 characters)

Trish

0	1	2	3	4	5
T	r	i	s	h	

A programmer can assign a string just as with other types. Ex: str1 = 'Hello', or str1 = str2. The input() function can also be used to get strings from the user.

An empty string is a sequence type with 0 elements, created with two quotes. Ex: `my_str = ''`.

zyDE 3.1.1: A program with strings.

Try the "Mad Libs" style game below.

Load default template...

```
#A 'Mad Libs' style game w
#verbs, etc., and then a s

#Get user's words
relative = input('Enter a
print()

food = input('Enter a type
print()

adjective = input('Enter a
print()
```

```
brother
burritos
macho
```

Run

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
print()

period = input('Enter a ti
print()

# Tell the story
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.1.2: String literals.

Indicate which items are string literals.

1) 'Hey'

- Yes
- No

2) 'Hey there.'

- Yes
- No

3) 674

- Yes
- No

4) '674'

- Yes
- No

5) "ok"

- Yes
- No

6) "a"

- Yes
- No

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.1.3: String basics.

1) Which answer creates a string variable first_name with a value 'Daniel'?

- `Daniel = first_name`
- `first_name = 'Daniel'`
- `first_name = Daniel`

2) Which answer prints the value of the first_name variable?

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

- `print(first_name)`
- `print('first_name')`
- `print("first_name")`

3) Which answer assigns first_name with a string read from input?

- `first_name = input`
- `input('Type your name:')`
- `first_name = input('Type your name:')`

4) Which answer assigns first_name with an empty string?

- `first_name =`
- `first_name = ''`
- `'' = first_name`

String length and indexing

A common operation is to find the length, or the number of characters, in a string. The ***len()*** built-in function can be used to find the length of a string (and any other sequence type).

Figure 3.1.1: Using len() to get the length of a string.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

The \ character after the string literal extends the string to the following line.

```
george_v = "His Majesty George V, by the Grace
of God, " \
           "of the United Kingdom of Great
Britain and " \
           "Ireland and of the British Dominions
beyond " \
           "the Seas, King, Defender of the
Faith, Emperor of India"
gandhi = 'Mohandas Karamchand Gandhi'
john_f_kennedy = 'JFK'

print(len(george_v), 'characters is much too
long of a name!')
print(len(gandhi), 'characters is better...')
print(len(john_f_kennedy), 'characters is short
enough.')
```

```
185 characters is much too
long of a name!
26 characters is better...
3 characters is short
enough.
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.1.4: Using len() to find the length of a string.

1) What is the length of the string "Santa"?

Check Show answer

2) Write a statement that prints the length of the string variable first_name.

Check Show answer

Programs commonly access an individual character of a string. As a sequence type, every character in a string has an index, or position, starting at 0 from the leftmost character. For example, the 'A' in string 'ABC' is at index 0, 'B' is at index 1, and 'C' is at index 2. A programmer can access a character at a specific index by appending ***brackets*** [ ] containing the index:

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Figure 3.1.2: Accessing individual characters of a string.

```
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print(alphabet[0], alphabet[1],
alphabet[25])
```

```
A B
Z
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Note that negative indices can be used to access characters starting from the right-most character of the string, instead of the leftmost. Ex: alphabet[-1] is 'Z'.

zyDE 3.1.2: String indexing.

Try the simple program that looks up the indices of letters in the alphabet. Try entering a negative value like -1, or -25.

Load default template...

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

user_number = int(input('Enter number to use as index: '))
print()

print('\nThe letter at index', user_number, 'of the alphabet

```

2

Run

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.1.5: String indexing.

1) What character is in index 2 of the string "America"?

Check Show answer

2) Write an expression that accesses the first character of the string my_country.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Check Show answer

3) Assign my_var with the last character in my_str. Use a negative index.

Check Show answer

Changing string variables and concatenating strings

Writing or altering individual characters of a string variable is not allowed. Strings are immutable objects, meaning that string values cannot change once created. Instead, an assignment statement must be used to update an entire string variable.

Figure 3.1.3: Strings are immutable and cannot be changed.

Individual characters of a string cannot be directly changed.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
alphabet =
'abcdefghijklmnopqrstuvwxyz'

# Change to upper case

alphabet[0] = 'A'   # Invalid: Cannot
change character
alphabet[1] = 'B'   # Invalid: Cannot
change character

print('Alphabet:', alphabet)
```

```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    alphabet[0] = 'A'   # Invalid:
Cannot change character
TypeError: 'str' object does not
support item assignment
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Instead, update the variable by assigning an entirely new string.

```
alphabet =
'abcdefghijklmnopqrstuvwxyz'

# Change to upper case
alphabet =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

print('Alphabet:', alphabet)
```

```
Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

A program can add new characters to the end of a string in a process known as ***string concatenation***. The expression `"New" + "York"` concatenates the strings New and York to create a new string NewYork. Most sequence types support concatenation. String concatenation does not contradict the immutability of strings, because the result of concatenation is a new string; the original strings are not altered.

Figure 3.1.4: String concatenation.

```
string_1 = 'abc'
string_2 = '123'
concatenated_string = string_1 +
string_2
print('Easy as ' +
concatenated_string)
```

```
Easy as
abc123
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.1.6: String variables.

1) Python string objects are mutable, meaning that individual characters can be changed.

- True
- False

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) Executing the statements:

```
address = '900 University Ave'
address[0] = '6'
address[1] = '2'
```

is a valid way to change address to '620 University Ave'.

- True
- False

3) Executing the statements:

```
address = '900 University Ave'
address = '620 University Ave'
```

is a valid way to change address to '620 University Ave'.

- True
- False

4) After the following executes, the value of address is '500 Floral Avenue'.

```
street_num = '500'
street = 'Floral Avenue'
address = street_num + ' ' +
street
```

- True
- False

CHALLENGE ACTIVITY 3.1.1: Enter the output of the string.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

Start

Type the program's output

```
print('Aerith')
```

```
Aerith
```

1 2 3 4 5 6

Check Next

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

CHALLENGE ACTIVITY 3.1.2: String basics.

566436.4601014.qx3zqy7

Start

Read strings student_name and course_name from input.

▶ **Click here for example**

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

print(student_name + ' is studying ' + course_name + '.')
```

1 2 3

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.2 String formatting

Formatted string literals (f-strings)

Program output includes expressions, such as variables or other calculations, as part of the output text. A ***formatted string literal***, or ***f-string***, allows a programmer to create a string with placeholder expressions that are evaluated as the program executes. An f-string starts with an `f` character before the starting quote and uses curly braces `{ }` to denote the placeholder expressions. A placeholder expression is also called a ***replacement field***, as its value replaces the expression in the final output.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.2.1: Creating literal strings with embedded expressions.

```
number = 6
amount = 32

print(f'{number} burritos cost ${amount}')
```

```
6 burritos cost $32
```

The {number} and {amount} expressions in the replacement fields are evaluated to their values of 6 and 32 in the final string.

Animation content:

The first expression is evaluated and the value of the number variable 6 is substituted into the string. The second expression is evaluated and the value of the amount variable 32 is substituted into the string. The resulting string literal is '6 burritos cost $32'.

Animation captions:

1. The first expression, {number}, is replaced with the value of the number variable.
2. The second expression, {amount}, is replaced with the value of the amount variable.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.2.2: Identify the output of f-strings.

Select the option that correctly prints the given output. Assume the following code is defined.

```
num_items = 3
cost_taco = 1.25
```

1) `I need 3 items please`

- `print(f'I need {num_items} please')`
- `print(f'{I need num_items items please}')`
- `print('I need {num_items} items please')`
- `print(f'I need {num_items} items please')`

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) `3 tacos cost 3.75`

- `print('{num_items} tacos cost {cost_taco * 3}')`
- `print(f'{num_items} tacos cost {cost_taco}')`
- `print(f'{num_items} tacos cost {cost_taco * num_items}')`
- `print(f'(num_items) tacos cost (cost_taco * num_items)')`

Additional f-string features

An = sign is provided after the expression in a replacement field to print both the expression and its result, which is a useful debugging technique when dynamically generating lots of strings and output. Ex: `f'{2*4=}'` produces the string "2*4=8".

Additionally, double braces `{{` and `}}` are used to place a curly brace into an f-string. Ex: `f'{{Jeff Bezos}}: Amazon'` produces the string "{Jeff Bezos}: Amazon".

Table 3.2.1: f-string examples.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Example	Output
`print(f'{2**2=}')`	`2**2=4`

`two_power_two = 2**2` `print(f'{two_power_two=}')`	`two_power_two=4`
`print(f'{2**2=},{2**4=}')`	`2**2=4,2**4=16`
`print(f'{{2**2}}')`	`{2**2}`
`print(f'{{{2**2=}}}')`	`{2**2=4}`

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.2.3: Output of f-strings using debug features and escape characters.

Enter the final value of `output`.

1)
```
output = f'{2*2}
```

Check Show answer

2)
```
output = f'{2*2=}'
```

Check Show answer

3)
```
kids = 4
adults = 2
output = f'{kids+adults}
total'
```

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

4)
```
kids = 4
adults = 2
output = f'{kids+adults=}
total'
```

Check Show answer

5) `output = f'{{2}} + {{3}} = {{5}}'`

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Format specifications

A ***format specification*** inside a replacement field allows a value's formatting in the string to be customized. Ex: Using a format specification, a variable with the integer value 4 can be output as a floating-point number (4.0), with leading zeros (004), and aligned to the left or right, etc.

A format specification is introduced with a colon `:` in the replacement field. The colon separates the "what" on the left from the "how" on the right. The left "what" side is an expression to be evaluated, perhaps just a variable name or a value. The right "how" side is the format specification that determines how to show that value using special characters. Ex: `{4:.2f}` formats `4` as `4.00`.

A ***presentation type*** is part of a format specification that determines how to represent a value in text form, such as an integer (4), a floating point (4.0), a fixed precision decimal (4.000), a percentage (4%), a binary (100), etc. A presentation type can be set in a replacement field by inserting a colon `:` and providing one of the presentation type characters described below.

More advanced format specifications, like fill and alignment, are provided in a later section.

Table 3.2.2: Common format specification presentation types.

Type	Description	Example	Output
s	String (default presentation type - can be omitted)	`name = "Aiden"` `print(f"{name:s}")`	`Aiden`
d	Decimal (integer values only)	`number = 4` `print(f"{number:d}")`	`4`
,d	Decimal (integer values only) with commas	`number = 7600` `print(f"{number:,d}")`	`7,600`

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

0[width]d	Leading 0 notation	`number = 4` `print(f"{number:03d}")`	`004`
b	Binary (integer values only)	`number = 4` `print(f"{number:b}")`	`100`
x, X	Hexadecimal in lowercase (x) and uppercase (X) (integer values only)	`number = 31` `print(f"{number:x}")`	`1f`
e	Exponent notation	`number = 44` `print(f"{number:e}")`	`4.400000e+01`
f	Fixed-point notation (six places of precision)	`number = 4` `print(f"{number:f}")`	`4.000000`
.[precision]f	Fixed-point notation (programmer-defined precision)	`number = 4` `print(f"{number:.2f}")`	`4.00`
,.[precision]f	Fixed-point notation (programmer-defined precision) with commas	`number = 7600.1` `print(f"{number:,.2f}")`	`7,600.10`

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.2.4: Format specifications and presentation types.

Enter the most appropriate format specification to produce the desired output.

1) The value of `num` as a decimal (base 10) integer: `31`

```
num = 31
print(f'{num:  }')
```

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) The value of `num` as a hexadecimal (base 16) integer: `1f`

```
num = 31
print(f'{num:    }')
```

Check Show answer

3) The value of `num` as a binary (base 2) integer: `11111`

```
num = 31
print(f'{num:    }')
```

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

CHALLENGE ACTIVITY 3.2.1: String formatting with f-strings.

Type the program's output.

Note: Each `print()` includes a newline, so create a newline after your output by pressing *Enter* or *Return* on your keyboard. No extra whitespace is needed.

566436.4601014.qx3zqy7

Start

Type the program's output

```
name = 'Ava'

print(f'Your name is {name}')
```

```
Your name is Ava
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 3.2.2: String formatting.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

Start

Select the most appropriate replacement field definition to generate the followir

The math class has 55 students.

num_students = 55

print(f'The math class has Pick students.')

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1 2 3

Check Next

CHALLENGE ACTIVITY 3.2.3: String formatting.

Start

color_1 and color_2 are read from input. Use an f-string to output the following on one line:

- 'Favorite colors: '
- The value of color_1
- ', '
- The value of color_2

▶ **Click here for example**

```
color_1 = input()
color_2 = input()

print(f''' Your code goes here ''')
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1 2

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.3 List basics

Creating a list

A ***container*** is a construct used to group related values together and contains references to other objects instead of data. A ***list*** is a container created by surrounding a sequence of variables or literals with brackets []. Ex: `my_list = [10, 'abc']` creates a new list variable my_list that contains the two items: 10 and 'abc'. A list item is called an ***element***. A list is mutable, meaning that the elements in a list can be replaced, reordered, or removed.

A list is also a sequence, meaning the contained elements are ordered by position in the list, known as the element's ***index***, starting with 0. `my_list = [ ]` creates an empty list.

The animation below shows how a list is created and managed by the interpreter. A list is an object, and its value is a sequence of references to the list's elements.

PARTICIPATION ACTIVITY 3.3.1: Creating lists.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Animation content:

Static figure: A Python command line and Python interpreter are displayed.
Step 1: User creates a new list. The line of code, prices = ['$20', 14.99, 5], appears in the Python command line. The line of code, prices = ['$20', 14.99, 5], appears in the Python interpreter and a new list labeled prices is displayed.
Step 2: The interpreter creates a new object for each list element. The list, prices, is populated with the elements:
'$20'
14.99
5
Step 3: 'prices' holds references to objects in the list. An arrow appears showing that prices[0] points to '$20', prices[1] points to 14.99, and prices[2] points to 5.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Animation captions:

1. User creates a new list.
2. The interpreter creates a new object for each list element.
3. 'prices' holds references to objects in the list.

zyDE 3.3.1: Creating lists.

The following program prints a list of names. Try adding your name to the list, and run the program again.

Load default template...

Run

```
names = ['Daniel', 'Roxanna',

print(names)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.3.2: Creating lists.

1) Write a statement that creates a list called my_nums, containing the elements 5, 10, and 20.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) Write a statement that creates a list called my_list with the elements -100 and the string 'lists are fun'.

Check Show answer

3) Write a statement that creates an empty list called class_grades.

Check Show answer

Accessing list elements

Lists are useful for reducing the number of variables in a program. Instead of having a separate variable for the name of every student in a class, or for every word in an email, a single list can store an entire collection of related variables.

Individual list elements can be accessed using an indexing expression by using brackets as in my_list[i], where i is an integer. This allows a programmer to quickly find the i'th element in a list.

A list's index must be an integer. The index cannot be a floating-point type, even if the value is a whole number like 0.0 or 1.0. Using any type besides an integer will produce a runtime error, and the program will terminate.

Figure 3.3.1: Access list elements using an indexing expression.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
# Some of the most expensive cars in the
world
lamborghini_veneno = 3900000  # $3.9 million!
bugatti_veyron = 2400000      # $2.4 million!
aston_martin_one77 = 1850000  # $1.85
million!

prices = [lamborghini_veneno, bugatti_veyron,
aston_martin_one77]

print(f'Lamborghini Veneno: {prices[0]}
dollars')
print(f'Bugatti Veyron Super Sport:
{prices[1]} dollars')
print(f'Aston Martin One-77: {prices[2]}
dollars')
```

```
Lamborghini Veneno: 3900000
dollars
Bugatti Veyron Super Sport:
2400000 dollars
Aston Martin One-77: 1850000
dollars
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Updating list elements

Lists are mutable, meaning that a programmer can change a list's contents. An element can be updated with a new value by performing an assignment to a position in the list.

Figure 3.3.2: Updating list elements.

```
my_nums = [5, 12, 20]
print(my_nums)

# Update a list
element
my_nums[1] = -28
print(my_nums)
```

```
[5, 12,
20]
[5, -28,
20]
```

PARTICIPATION ACTIVITY 3.3.3: Accessing and updating list elements.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1) Write a statement that assigns my_var with the 3rd element of my_list.

Check Show answer

2) Write a statement that assigns the 2nd element of my_towns with 'Detroit'.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Adding and removing list elements

Since lists are mutable, a programmer can also use methods to add and remove elements. A ***method*** instructs an object to perform some action, and is executed by specifying the method name following a "." symbol and an object. The ***append()*** list method is used to add new elements to a list. Elements can be removed using the ***pop()*** or ***remove()*** methods. Methods are covered in greater detail in another section.

Adding elements to a list:

- list.append(value): Adds value to the end of list. Ex: `my_list.append('abc')`

Removing elements from a list:

- list.pop(i): Removes the element at index i from list. Ex: `my_list.pop(1)`
- list.remove(v): Removes the first element whose value is v. Ex: `my_list.remove('abc')`

PARTICIPATION ACTIVITY 3.3.4: Adding and removing list elements.

```
my_list = [10, 'bw']
print(my_list)

my_list.append('abc')
print(f'After append: {my_list}')

my_list.pop(1)
print(f'After pop: {my_list}')

my_list.remove('abc')
print(f'After remove: {my_list}')
```

my_list:

10

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
[10, 'bw']
After append: [10, 'bw', 'abc']
After pop: [10, 'abc']
After remove: [10]
```

Animation content:

Static figure: A code block, a list labeled my_list containing the elements 10 and 'bw', and an output console with the text, [10, 'bw'], are displayed.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Begin Python code:

```
my_list = [10, 'bw']
print(my_list)

my_list.append('abc')
print(f'After append: {my_list}')

my_list.pop(1)
print(f'After pop: {my_list}')

my_list.remove('abc')
print(f'After remove: {my_list}')
```

End Python code.

Step 1: append() adds an element to the end of the list. The line of code, my_list.append('abc'), is highlighted and the element 'abc' is added to end of my_list. The line of code, print(f'After append: {my_list}'), is highlighted and the text, After append: [10, 'bw', 'abc'], is displayed in the output console.

Step 2: pop() removes the element at the given index from the list. 'bw', which is at index 1, is removed and 'abc' is now at index 1. The line of code, my_list.pop(1), is highlighted and the element 'bw' is removed from my_list. The line of code, print(f'After pop: {my_list}'), is highlighted and the text, After pop: [10, 'abc'], is displayed in the output console.

Step 3: remove() removes the first element with a given value. 'abc' is removed and now the list only has one element. The line of code, my_list.remove('abc'), is highlighted and the element 'abc' is removed from my_list. The line of code, print(f'After remove: {my_list}'), is highlighted and the text, After remove: [10], is displayed in the output console.

Animation captions:

1. append() adds an element to the end of the list.
2. pop() removes the element at the given index from the list. 'bw', which is at index 1, is removed and 'abc' is now at index 1.
3. remove() removes the first element with a given value. 'abc' is removed and now the list only has one element.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main

PARTICIPATION ACTIVITY 3.3.5: List modification.

Write a statement that performs the desired action. Assume the list `house_prices = ['$140,000', '$550,000', '$480,000']` exists.

1) Update the price of the second item in house_prices to '$175,000'.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) Add a price to the end of the list with a value of '$1,000,000'.

Check Show answer

3) Remove the first element from house_prices, using the pop() method.

Check Show answer

4) Remove '$140,000' from house_prices, using the remove() method.

Check Show answer

Sequence-type methods and functions

Sequence-type functions are built-in functions that operate on sequences like lists and strings. ***Sequence-type methods*** are methods built into the class definitions of sequences like lists and strings. A subset of such functions and methods is provided below.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Table 3.3.1: Some of the functions and methods useful to lists.

Operation	Description

len(list)	Find the length of the list.
list1 + list2	Produce a new list by concatenating list2 to the end of list1.
min(list)	Find the element in the list with the smallest value. All elements must be of the same type.
max(list)	Find the element in the list with the largest value. All elements must be of the same type.
sum(list)	Find the sum of all elements of a list (numbers only).
list.index(val)	Find the index of the first element in the list whose value matches val.
list.count(val)	Count the number of occurrences of the value val in the list.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Figure 3.3.3: Using sequence-type functions with lists.

```
# Concatenating lists
house_prices = [380000, 900000, 875000] +
[225000]
print(f'There are {len(house_prices)} prices in
the list.')

# Finding min, max
print(f'Cheapest house: {min(house_prices)}')
print(f'Most expensive house:
{max(house_prices)}')
```

```
There are 4 prices in the
list.
Cheapest house: 225000
Most expensive house:
900000
```

Note that lists can contain mixed types of objects. Ex: `x = [1, 2.5, 'abc']` creates a new list x that contains an integer, a floating-point number, and a string. Later material explores lists in detail, including how lists can even contain other lists as elements.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

zyDE 3.3.2: Student grade statistics.

The following program calculates some information regarding final and midterm scores. Try enhancing the program by calculating the average midterm and final scores.

Load default template...

```
#Program to calculate statistics from student test scores.
midterm_scores = [99.5, 78.25, 76, 58.5, 100, 87.5, 91, 68
final_scores = [55, 62, 100, 98.75, 80, 76.5, 85.25]

#Combine the scores into a single list
all_scores = midterm_scores + final_scores

num_midterm_scores = len(midterm_scores)
num_final_scores = len(final_scores)

print(f'{num_midterm_scores} students took the midterm.')
print(f'{num_final_scores} students took the final.')

#Calculate the number of students who took the midterm but
dropped_students = num_midterm_scores - num_final_scores
print(f'{dropped_students} students must have dropped the

lowest_final = min(final_scores)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Run

PARTICIPATION ACTIVITY 3.3.6: Using sequence-type functions.

1) Write an expression that concatenates the list feb_temps to the end of jan_temps.

Check Show answer

2) Write an expression that finds the minimum value in the list total_prices.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3) Write a statement that assigns the variable avg_price with the average of the elements of prices.

Check Show answer

CHALLENGE ACTIVITY 3.3.1: List functions and methods.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

Start

user_ages = [7, 9, 2]

What is the value of len(user_ages)?

Select

1	2	3	4	5

Check Next

CHALLENGE ACTIVITY 3.3.2: List basics.

566436.4601014.qx3zqy7

Start

Two names are read from input into variables name1 and name2. Then, two ages are read from input i
age2. Create a list named names_list to hold name1 and name2, and create a list named ages_list to h

▶ Click here to show example

```
name1 = input()
name2 = input()
age1 = int(input())
age2 = int(input())

''' Your code goes here '''

print(names_list)
print(ages_list)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1 2 3

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.4 Tuple basics

Tuples

A ***tuple***, usually pronounced "tuhple" or "toople," stores a collection of data, like a list, but is immutable – once created, the tuple's elements cannot be changed. A tuple is also a sequence type, supporting len(), indexing, and other sequence functions. A new tuple is generated by creating a list of comma-separated values, such as `5, 15, 20`. Typically, tuples are surrounded with parentheses, as in `(5, 15, 20)`. Note that printing a tuple always displays surrounding parentheses.

A tuple is not as common as a list in practical usage but can be useful when a programmer wants to ensure that values do not change. Tuples are typically used when element position, and not just the relative ordering of elements, is important. Ex: A tuple might store the latitude and longitude of a landmark because a programmer knows that the first element should be the latitude, the second element should be the longitude, and the landmark will never move from those coordinates.

Figure 3.4.1: Using tuples.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
white_house_coordinates = (38.8977,
77.0366)
print(f'Coordinates:
{white_house_coordinates}')
print(f'Tuple length:
{len(white_house_coordinates)}')

# Access tuples via index
print(f'\nLatitude:
{white_house_coordinates[0]} north')
print(f'Longitude:
{white_house_coordinates[1]} west\n')

# Error. Tuples are immutable
white_house_coordinates[1] = 50
```

```
Coordinates: (38.8977, 77.0366)
Tuple length: 2

Latitude: 38.8977 north
Longitude: 77.0366 west

Traceback (most recent call
last):
  File "<stdin>", line 10, in
<module>
TypeError: 'tuple' object does
not support item assignment
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.4.1: Tuples.

1) Create a new variable `point` that is a tuple containing the strings `'X string'` and `'Y string'`.

Check Show answer

2) If the value of variable `friends` is the tuple `('Cleopatra', 'Marc', 'Seneca')`, then what is the result of `len(friends)`?

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Named tuples

A program commonly captures collections of data. Ex: a car is described using a series of variables describing the make, model, retail price, horsepower, and number of seats. A ***named tuple*** allows the programmer to define a new simple data type that consists of named attributes. A `Car` named tuple with fields like `Car.price` and `Car.horsepower` would more clearly represent a car object than a list with index positions correlating to some attributes.

The ***namedtuple*** container must be imported to create a new named tuple. Once the container is imported, the named tuple should be created like in the example below, where the name and attribute names of the named tuple are provided as arguments to the namedtuple constructor. Note that the fields to include in the named tuple are found in a list, but the fields may also be a single string with space- or comma-separated values.

Figure 3.4.2: Creating named tuples.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
from collections import namedtuple

Car = namedtuple('Car', ['make','model','price','horsepower','seats'])  # Create the named tuple

chevy_blazer = Car('Chevrolet', 'Blazer', 32000, 275, 8)  # Use the named tuple to describe a car
chevy_impala = Car('Chevrolet', 'Impala', 37495, 305, 5)  # Use the named tuple to describe a different car

print(chevy_blazer)
print(chevy_impala)
```

```
Car(make='Chevrolet', model='Blazer', price=32000, horsepower=275, seats=8)
Car(make='Chevrolet', model='Impala', price=37495, horsepower=305, seats=5)
```

`namedtuple()` creates only the new simple data type and does not create new data objects. Above, a new data object is not created until `Car()` is called with appropriate values. A data object's attributes can be accessed using dot notation, as in `chevy_blazer.price`. This "named" attribute is simpler to read than a list or tuple referenced via index such as `chevy_blazer[2]`.

Like normal tuples, named tuples are immutable. A programmer wishing to edit a named tuple would replace the named tuple with a new object.

PARTICIPATION ACTIVITY 3.4.2: Named tuples.

Assume namedtuple has been imported. Use a list of strings in the `namedtuple()` constructor where applicable.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1) Complete the following named tuple definition that describes a house.

```
House =
('House', ['street',
'postal_code', 'country'])
```

Check Show answer

2) Create a new named tuple `Dog` that has the attributes `name`, `breed`, and `color`.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3) Let `Address = namedtuple('Address', ['street', 'city', 'country'])`. Create a new address object `house` where `house.street` is "221B Baker Street", `house.city` is "London", and `house.country` is "England".

Check Show answer

4) Given the following named tuple `Car = namedtuple('Car', ['make', 'model', 'price', 'horsepower', 'seats'])`, and data objects `car1` and `car2`, write an expression that computes the sum of the price of both cars.

Check Show answer

CHALLENGE ACTIVITY 3.4.1: Tuple basics.

566436.4601014.qx3zqy7

Start

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

City is a named tuple with fields: name, state, and population. Variable city is a City object created with integer read from input as attributes. Output the following:

- "City name: " followed by city's name attribute
- "State: " followed by city's state attribute
- "Population: " followed by city's population attribute

End each output with a newline.

▶ **Click here for example**

```
from collections import namedtuple

City = namedtuple('City', ['name', 'state', 'population'])

city = City(input(), input(), int(input()))

''' Your code goes here '''

```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1 2 3

Check Next level

3.5 Set basics

Set basics

A ***set*** is an unordered collection of unique elements. A set has the following properties:

- Elements are unordered: Elements in the set do not have a position or index.
- Elements are unique: No elements in the set share the same value.

A set can be created using the ***set()*** function, which accepts a sequence-type iterable object (list, tuple, string, etc.) whose elements are inserted into the set. A ***set literal*** can be written using curly braces `{ }` with commas separating set elements. Note that an empty set can only be created using `set()`.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Figure 3.5.1: Creating sets.

```
# Create a set using the set()
function.
nums1 = set([1, 2, 3])

# Create a set using a set literal.
nums2 = { 7, 8, 9 }

# Print the contents of the sets.
print(nums1)
print(nums2)
```

```
{1, 2,
3}
{7, 8,
9}
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Because the elements of a set are unordered and have no meaningful position in the collection, the index operator is not valid. Attempting to access the element of a set by position, for example `nums1[2]` to access the element at index 2, is invalid and will produce a runtime error.

A set is often used to reduce a list of items that potentially contains duplicates into a collection of unique values. Simply passing a list into `set()` will cause any duplicates to be omitted in the created set.

zyDE 3.5.1: Creating sets.

Load default template...

Run

```
# Initial list contains some
first_names = [ 'Alba', 'Hema

# Creating a set removes any
names_set = set(first_names)

print(names_set)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.5.1: Basic sets.

1) What's the result of `set(['A', 'Z'])`?

- A set that contains `'A'` and `'Z'`.
- A list with the following elements: `['A', 'Z']`.
- Error: invalid syntax.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) What's the result of `set(10, 20, 25)`?

- A list with the following elements: `[10, 20, 25]`.
- A set that contains `10`, `20`, and `25`.
- Error: invalid syntax.

3) What's the result of `set([100, 200, 100, 200, 300])`?

- A list with the following elements: `[100, 200, 100, 200, 300]`.
- A set that contains `100`, `200`, and `300`.
- A set that contains `100`, `200`, `300`, another `100`, and another `200`.
- Error: invalid syntax.

Modifying sets

Sets are mutable – elements can be added or removed using set methods. The ***add()*** method places a new element into the set if the set does not contain an element with the provided value. The ***remove()*** and ***pop()*** methods remove an element from the set.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Additionally, sets support the `len()` function to return the number of elements in a set. To check if a specific value exists in a set, a membership test such as `value in set` (discussed in another section) can be used.

Adding elements to a set:

- set.add(value): Add value into the set. Ex: `my_set.add('abc')`

Remove elements from a set:

- set.remove(value): Remove the element with given value from the set. Raises KeyError if value is not found. Ex: `my_set.remove('abc')`
- set.pop(): Remove a random element from the set. Ex: `my_set.pop()`

Table 3.5.1: Some of the methods useful to sets.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Operation	Description
len(set)	Find the length (number of elements) of the set.
set1.update(set2)	Adds the elements in set2 to set1.
set.add(value)	Adds value into the set.
set.remove(value)	Removes value from the set. Raises KeyError if value is not found.
set.pop()	Removes a random element from the set.
set.clear()	Clears all elements from the set.

PARTICIPATION ACTIVITY 3.5.2: Modifying sets.

```
# Create sets
names1 = {'Pedro', 'Khan', 'Dean'}
names2 = {'Emilia', 'Kara', 'Tia'}

# Add element to set
names1.add('Hyungu')

# Add names2 to names1
names1.update(names2)

# Remove element from set
names1.remove('Dean')

# Clear all elements from set
names2.clear()
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Animation content:

Static figure: A code block is displayed.
Begin Python code:

```
# Create sets
names1 = {'Pedro', 'Khan', 'Dean'}
names2 = {'Emilia', 'Kara', 'Tia'}

# Add element to set
names1.add('Hyungu')

# Add names2 to names1
names1.update(names2)

# Remove element from set
names1.remove('Dean')

# Clear all elements from set
names2.clear()
```

End Python code.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Step 1: Sets can be created using braces {} with commas separating the elements. The line of code, names1 = {'Pedro', 'Khan', 'Dean'}, is highlighted and a set labeled names1 appears with the elements:
Khan
Dean
Pedro
The line of code, names2 = {'Emilia', 'Kara', 'Tia'}, is highlighted and a set labeled names2 appears with the elements:
Tia
Kara
Emilia
Step 2: The add() method adds a single element to a set. The line of code, names1.add('Hyungu'), is highlighted and the element, Hyungu, is added to the end of the names1 set.
Step 3: The update() method adds the elements of one set to another set. The line of code, names1.update(names2), is highlighted and the names1 set now contains:
Khan
Dean
Pedro
Hyungu
Tia
Kara
Emilia

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Step 4: The remove() method removes a single element from a set. The line of code, names1.remove('Dean'), is highlighted and removes the element, Dean, from the names1 set.
Step 5: The clear() method removes all elements from a set, leaving the set with a length of 0. The line of code, names2.clear(), is highlighted and all elements are removed from the names2 set.

Animation captions:

1. Sets can be created using braces {} with commas separating the elements.
2. The add() method adds a single element to a set.
3. The update() method adds the elements of one set to another set.
4. The remove() method removes a single element from a set.
5. The clear() method removes all elements from a set, leaving the set with a length of 0.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.5.3: Modifying sets.

Write a line of code to complete the following operations.

1) Add the literal `'Ryder'` to the set `names`.

Check Show answer

2) Add all of the elements of set `goblins` into set `monsters`.

Check Show answer

3) Remove all of the elements from the `trolls` set.

Check Show answer

4) Get the number of elements in the set `elves`.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Set operations

Python set objects support typical set theory operations such as intersections and unions. Provided below is a brief overview of common set operations supported in Python:

Table 3.5.2: Common set theory operations.

.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Operation	Description
set.intersection(set_a, set_b, set_c...)	Returns a new set containing only the elements in common between `set` and all provided sets.
set.union(set_a, set_b, set_c...)	Returns a new set containing all of the unique elements in all sets.
set.difference(set_a, set_b, set_c...)	Returns a set containing only the elements of `set` that are not found in any of the provided sets.
set_a.symmetric_difference(set_b)	Returns a set containing only elements that appear in exactly one of `set_a` or `set_b`

PARTICIPATION ACTIVITY 3.5.4: Set theory operations.

```
# Create sets
names1 = {'Corrin', 'Pedro', 'Khan', 'Dean'}
names2 = {'Emilia', 'Kara', 'Corrin', 'Tia'}
names3 = {'Karat', 'Kara', 'Karah', 'Khan'}
names4 = {'Khan', 'Dean', 'Pascale'}

# Union names1 and names2
result_set = names1.union(names2)

# Intersection btwn result_set and names3
result_set = result_set.intersection(names3)

# Difference btwn result_set and names4
result_set = result_set.difference(names4)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Animation content:

Static figure: A code block is displayed.
Begin Python code:

```
# Create sets
names1 = {'Corrin', 'Pedro', 'Khan', 'Dean'}
names2 = {'Emilia', 'Kara', 'Corrin', 'Tia'}
names3 = {'Karat', 'Kara', 'Karah', 'Khan'}
names4 = {'Khan', 'Dean', 'Pascale'}

# Union names1 and names2
result_set = names1.union(names2)

# Intersection btwn result_set and names3
result_set = result_set.intersection(names3)

# Difference btwn result_set and names4
result_set = result_set.difference(names4)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

End Python code.

A set labeled names1 is displayed with the elements:
Khan
Dean
Pedro
Corrin
A set labeled names2 is displayed with the elements:
Tia
Kara
Corrin
Emilia
A set labeled names3 is displayed with the elements:
Kara
Karah
Karat
Khan
A set labeled names4 is displayed with the elements:
Pascale
Khan
Dean

Step 1: The union() method builds a set containing the unique elements from names1 and names2. 'Corrin' only appears once in the resulting set. The line of code, result_set = names1.union(names2), is highlighted and a new set labeled, result_set, is created containing the elements:

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Khan
Dean
Pedro
Corrin
Tia
Kara
Emilia

Step 2: The intersection() method builds a set that contains all common elements between result_set and names3. The line of code, result_set = result_set.intersection(names3), is highlighted and the set, result_set, now contains the elements:
Khan
Kara
Step 3: The difference() method builds a set that contains elements only found in result_set that are not in names4. The line of code, result_set = result_set.difference(names4), is highlighted and the set, result_set, now only contains the element:
Kara

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Animation captions:

1. The union() method builds a set containing the unique elements from names1 and names2. 'Corrin' only appears once in the resulting set.
2. The intersection() method builds a set that contains all common elements between result_set and names3.
3. The difference() method builds a set that contains elements only found in result_set that are not in names4.

PARTICIPATION ACTIVITY 3.5.5: Set theory operations.

Assume that:

- `monsters = {'Gorgon', 'Medusa'}`
- `trolls = {'William', 'Bert', 'Tom'}`
- `horde = {'Gorgon', 'Bert', 'Tom'}`

Fill in the code to complete the line that would produce the given set.

1) `{'Gorgon', 'Bert', 'Tom', 'Medusa', 'William'}`

```
monsters.[    ](trolls)
```

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) `{'Gorgon'}`

```
monsters.[    ](horde)
```

Check Show answer

3) `{'Medusa', 'Bert', 'Tom'}`

```
monsters.symmetric_difference(
)
```

Check Show answer

CHALLENGE ACTIVITY 3.5.1: Set basics.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

Start

Variables animal1, animal2, animal3, and animal4 are read from input. Create a set named unique_anim
animal2, animal3, and animal4.

▶ **Click here for example**

Note: Because sets are unordered, the set is printed using the sorted() function here for comparison.

```
animal1 = input()
animal2 = input()
animal3 = input()
animal4 = input()

''' Your code goes here '''

print(sorted(unique_animals))
```

1 2 3

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.6 Dictionary basics

Creating a dictionary

Consider a normal English language dictionary. A reader looks up the word "cat" and finds the definition, "A small, domesticated carnivore." The relationship between "cat" and its definition is *associative*, i.e., "cat" is associated with words describing "cat."

A ***dictionary*** is a Python container used to describe associative relationships. A dictionary is represented by the ***dict*** object type. A dictionary associates (or "maps") keys with values. A ***key*** is a term that can be located in a dictionary, such as the word "cat" in the English dictionary. A ***value*** describes some data associated with a key, such as a definition. A key can be any immutable type, such as a number, string, or tuple; a value can be any type.

©zyBooks 11/21/24 19:16 2360507
Liz Vogiatzis Main
KVCCCIS216JohnsonFall2024

A dict object is created using ***curly braces*** { } to surround the ***key:value pairs*** that comprise the dictionary contents. Ex: `players = {'Lionel Messi': 10, 'Cristiano Ronaldo': 7}` creates a dictionary called players with two keys: 'Lionel Messi' and 'Cristiano Ronaldo', associated with the values 10 and 7 (their respective jersey numbers). An empty dictionary is created with the expression `players = { }`.

Dictionaries are typically used in place of lists when an associative relationship exists. Ex: If a program contains a collection of anonymous student test scores, those scores should be stored in a list. However, if each score is associated with a student name, a dictionary could be used to associate student names to their score. Other examples of associative relationships include last names and addresses, car models and price, or student ID number and university email address.

Figure 3.6.1: Creating a dictionary.

```
players = {
    'Lionel Messi': 10,
    'Cristiano Ronaldo':
7
}

print(players)
```

```
{'Lionel Messi': 10, 'Cristiano Ronaldo':
7}
```

Note that formatting list or dictionary entries like in the above example, where elements appear on consecutive lines, helps to improve the readability of the code. The behavior of the code is not changed.

©zyBooks 11/21/24 19:16 2360507
Liz Vogiatzis Main
KVCCCIS216JohnsonFall2024

zyDE 3.6.1: Creating dictionaries.

Run the program below that displays the caffeine content in milligrams for 100 ml/grams of some popular foods. The indentation and spacing of the caffeine_content_mg key-value pairs simply provides more

readability. Note that order *is* maintained in the dict when printed (not standard before Python 3.7).

Try adding new items into the dictionary, using this U.S. federal government report on caffeine content.

Load default template... Run

```
caffeine_content_mg = {
    'Mr. Goodbar chocolate':
    'Red Bull': 33,
    'Monster Hitman Sniper e
    'Lipton Brisk iced tea -
    'dark chocolate coated c
    'Regular drip or percola
    'Buzz Bites Chocolate Ch
}

print(caffeine_content_mg)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.6.1: Create a dictionary.

1) Use braces to create a dictionary called ages that maps the names 'Bob' and 'Frank' to their ages, 27 and 75, respectively. For this exercise, make 'Bob' the first entry in the dict.

ages =

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Accessing dictionary entries

Though dictionaries maintain a left-to-right ordering, dictionary entries cannot be accessed by indexing. To access an entry, the key is specified in brackets [ ]. If no entry with a matching key exists in the dictionary, then a ***KeyError*** runtime error occurs and the program is terminated.

Figure 3.6.2: Accessing dictionary entries.

```
prices = {'apples': 1.99, 'oranges': 1.49}

print(f'The price of apples is
{prices["apples"]}')
print(f'\nThe price of lemons is
{prices["lemons"]}')
```

```
The price of apples is 1.99
Traceback (most recent call
last):
  File "<stdin>", line 3, in
<module>
KeyError: 'lemons'
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.6.2: Accessing dictionary entries.

1) A dictionary entry is accessed by placing a key in curly braces { }.

- True
- False

2) Dictionary entries are ordered by position.

- True
- False

Adding, modifying, and removing dictionary entries

A dictionary is mutable, so entries can be added, modified, and deleted as necessary by a programmer. A new dictionary entry is added by using brackets to specify the key: `prices['banana'] = 1.49`. A dictionary key is unique; attempting to create a new entry with a key that already exists in the dictionary *replaces* the existing entry. The ***del*** keyword is used to remove entries from a dictionary: `del prices['papaya']` removes the entry whose key is 'papaya'. If the requested key to delete does not exist, then a KeyError occurs.

Adding new entries to a dictionary:

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

- dict[k] = v: Adds the new key-value pair k-v, if dict[k] does not already exist.
  Example: `students['John'] = 'A+'`

Modifying existing entries in a dictionary:

- dict[k] = v: Updates the existing entry dict[k], if dict[k] already exists.
  Example: `students['Jessica'] = 'A+'`

Removing entries from a dictionary:

- del dict[k]: Deletes the entry dict[k].
  Example: `del students['Rachel']`

Figure 3.6.3: Adding and editing dictionary entries.

```
prices = {}  # Create empty dictionary
prices['banana'] = 1.49  # Add new entry
print(prices)

prices['banana'] = 1.69  # Modify entry
print(prices)

del prices['banana']  # Remove entry
print(prices)
```

```
{'banana': 1.49}
{'banana': 1.69}
{}
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.6.3: Modifying dictionaries.

1) Which statement adds 'pears' to the following dictionary?

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
```

- prices['pears'] = 1.79
- prices['pears': 1.79]

2) Executing the following statements produces a KeyError:

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
del prices['limes']
```

- True
- False

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3) Executing the following statements adds a new entry to the dictionary:

```
prices = {'apples': 1.99, 'oranges': 1.49, 'kiwi': 0.79}
prices['oranges'] = 1.29
```

- True

False

CHALLENGE ACTIVITY | 3.6.1: Dictionary basics.

566436.4601014.qx3zqy7

Start

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Four integers are read from input into variables card1, card2, card3, and card4. Create a dictionary nam containing the following key-value pairs:

- Key 'Hearts' with the value card1
- Key 'Diamonds' with the value card2
- Key 'Clubs' with the value card3
- Key 'Spades' with the value card4

▶ **Click here for example**

```
card1 = int(input())
card2 = int(input())
card3 = int(input())
card4 = int(input())

''' Your code goes here '''

print(f"Hearts: {cards_in_hand['Hearts']}")
print(f"Diamonds: {cards_in_hand['Diamonds']}")
print(f"Clubs: {cards_in_hand['Clubs']}")
print(f"Spades: {cards_in_hand['Spades']}")
```

1 2 3

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.7 Common data types summary

The most common Python types are presented below.

Common data types

Numeric types int and float represent the most common types used to store data. All numeric types support the normal mathematical operations such as addition, subtraction, multiplication, and division, among others.

Table 3.7.1: Common numeric data types.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Type	Notes
int	Numeric type: Used for variable-width integers.
float	Numeric type: Used for floating-point numbers.

Sequence types string, list, and tuple are containers for collections of objects ordered by position in the sequence, where the first object has an index of 0 and subsequent elements have indices 1, 2, etc. A list and a tuple are very similar, except that a list is mutable and individual elements may be edited or removed. Conversely, a tuple is immutable and individual elements may not be edited or removed. Lists and tuples can contain any type, whereas a string contains only single characters. A programmer can apply sequence-type functions such as len() and element indexing using brackets [ ] to any sequence type.

The only ***mapping type*** in Python is the dict type. Like a sequence type, a dict serves as a container. However, each element of a dict is independent, having no special ordering or relation to other elements. A dictionary uses key-value pairs to associate a key with a value.

Table 3.7.2: Containers: sequence and mapping types.

Type	Notes
string	Sequence type: Used for text.
list	Sequence type: A mutable container with ordered elements.
tuple	Sequence type: An immutable container with ordered elements.
set	Set type: A mutable container with unordered and unique elements.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

dict	Mapping type: A container with key-values associated elements.

PARTICIPATION ACTIVITY 3.7.1: Common data types.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1) The list `['a', 'b', 3]` is invalid because the list contains a mix of strings and integers.

- True
- False

2) int and float types can always hold the exact same values.

- True
- False

3) A sorted collection of integers might best be contained in a list.

- True
- False

Choosing a container type

New programmers often struggle with choosing a container type that best fits their needs, such as a list, tuple, or dict. In general, a programmer might use a list when data has an order, such as lines of text on a page. A programmer might use a tuple instead of a list if the contained data should not change. If order is not important, a programmer might use a dictionary to capture relationships between elements, such as student names and grades.

PARTICIPATION ACTIVITY 3.7.2: Choosing among different container types.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Choose the container that best fits the described data.

1) Student test scores that may later be adjusted, ordered from best to worst.

- list
- tuple

- dict

2) A single student's name and their final grade in the class.

- list
- tuple
- dict

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3) Names and current grades for all students in the class.

- list
- tuple
- dict

PARTICIPATION ACTIVITY 3.7.3: Finding errors in container code.

Click on the error.

1)

```
# Student grade program

students = ['Jo', 'Bob',
            'Amy']

grades = {}

# Get student name, grade
name = input('name:')
grade = input('grade:')# Assign grade

grades.append(name) = grade
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2)

```
workers = ('Jo', 'Amy')

# Remove Amy from workers
```

```
del workers[1]

# Print worker at index 0

print(f'Jo: {workers[0]}')
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.8 Additional practice: Grade calculation

The following program calculates an overall grade in a course based on three equally weighted exams.

zyDE 3.8.1: Grade calculator: Average score on three exams.

Load default template...

```
exam1_grade = float(input('Enter score on Exam 1 (out of 100)
exam2_grade = float(input('Enter score on Exam 2 (out of 100)
exam3_grade = float(input('Enter score on Exam 3 (out of 100)

overall_grade = (exam1_grade + exam2_grade + exam3_grade) / 3

print(f'Your overall grade is: {overall_grade}')
```

```
70
75
91
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Run

Create a different version of the program that:

1. Calculates the overall grade for four equally weighted programming assignments, in which each assignment is graded out of 50 points. Hint: First calculate the percentage for each assignment (e.g., score / 50), then calculate the overall grade percentage (be sure to multiply the result by 100).
2. Calculates the overall grade for four equally weighted programming assignments, in which assignments 1 and 2 are graded out of 50 points and assignments 3 and 4 are graded out of 75 points.
3. Calculates the overall grade for a course with three equally weighted exams (graded out of 100 points) that account for 60% of the overall grade and four equally weighted programming assignments (graded out of 50 points) that account for 40% of the overall grade. Hint: The overall grade can be calculated as 0.6 * averageExamScore + 0.4 * averageProgScore.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.9 Type conversions

Type conversions

A calculation sometimes must mix integer and floating-point numbers. For example, given that about 50.4% of human births are males, then `0.504 * num_births` calculates the number of expected males in num_births. If num_births is an integer type, then the expression combines a floating point and integer.

A ***type conversion*** is a conversion of one type to another, such as an integer to a float. An ***implicit conversion*** is a type conversion automatically made by the interpreter, usually between numeric types. For example, the result of an arithmetic operation like + or * will be a float only if either operand of the operation is a float.

- `1 + 2` returns an integer type.
- `1 + 2.0` returns a float type.
- `1.0 + 2.0` returns a float type.

integer-to-float conversion is straightforward: 25 becomes 25.0.

float-to-integer conversion just drops the fraction: 4.9 becomes 4.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.9.1: Implicit conversions between float and integer.

Type the value held in the variable after the assignment statement, given:

- num_items = 5
- item_weight = 0.5

For any floating-point answer, type answer to tenths. Ex: 8.0, 6.5, or 0.1

1) num_items + num_items

Check Show answer

2) item_weight * num_items

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Check Show answer

3) (num_items + num_items) * item_weight

Check Show answer

Conversion functions

Sometimes a programmer needs to explicitly convert an item's type. Conversion can be explicitly performed using the conversion functions below:

Table 3.9.1: Conversion functions for some common types.

Function	Notes	Can convert:
int()	Creates integers	integer, float, strings w/integers only
float()	Creates floats	integer, float, strings w/integers or fractions
str()	Creates strings	Any

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Converting a float to an integer will truncate the floating-point number's fraction. For example, the variable temperature might have a value of 18.75232, but can be converted to an integer expression `int(temperature)`. The result would have the value 18 with the fractional part removed.

Conversion of types is very common. In fact, all user input obtained using input() is initially a string, and a programmer must explicitly convert the input to a numeric type.

Strings can also be converted to numeric types, if the strings follow the correct formatting, i.e. using only numbers and possibly a decimal point. For example, `int('500')` yields an integer with a value of 500, and `float('1.75')` yields the floating-point value 1.75.

zyDE 3.9.1: Simple example of converting float and integer types.

Run the below program. Observe how the type conversion affects the entered number. Change the input to 18.552 and run the program again.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Load default template...

```
input_text = input('Enter a n
float_variable = float(input_
int_variable = int(float_vari

print(f'original input text:
print(f'input text converted
print(f'float variable conver

```

18

Run

PARTICIPATION ACTIVITY 3.9.2: Type conversions.

What is the result of each expression?

1) int(1.55)

- 1.55
- 1
- '1.55'

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) float("7.99")

- 7.0
- 8.0
- 7.99

3) str(99)

- 99
- 99.0
- '99'

CHALLENGE ACTIVITY 3.9.1: Enter the output of the conversion expression.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

Start

Type the program's output

```
number = 1
new_number = number * 3
print(new_number)
```

3

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 3.9.2: Type conversions.

566436.4601014.qx3zqy7

Start

A string is read from input into variable num_miles. Use a conversion function to convert the value of nu
assign the value to the variable int_miles.

▶ **Click here for example**

```
num_miles = input()

''' Your code goes here '''

print(int_miles + 14)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

1 2

Check Next level

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.10 Binary numbers

Binary numbers

Normally, a programmer thinks in terms of base 10 numbers. However, a computer must allocate some finite quantity of bits (e.g., 32 bits) for a variable, and that quantity of bits limits the range of numbers that the variable can represent. Python allocates additional memory to accommodate numbers of large sizes (past a typical 32 or 64 bit size), and a Python programmer need not think of such low-level details. However, binary base computation is an important part of computer science, so some background on how the quantity of bits influences a variable's number range is helpful.

Because each memory location is composed of bits (0s and 1s), a processor stores a number using base 2, known as a ***binary number***.

For a number in the more familiar base 10, known as a ***decimal number***, each digit must be 0-9, and each digit's place is weighed by increasing powers of 10.

Table 3.10.1: Decimal numbers use weighted powers of 10.

Decimal number with three digits	Representation
212	$= 2 \cdot 10^2 \quad + 1 \cdot 10^1 \quad + 2 \cdot 10^0$ $= 2 \cdot 100 \quad + 1 \cdot 10 \quad + 2 \cdot 1$ $= 200 \quad + 10 \quad + 2$ $= 212$

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

In ***base 2***, each digit must be 0-1 and each digit's place is weighted by increasing powers of 2.

Table 3.10.2: Binary numbers use weighted powers of 2.

Binary number with 4 bits	Representation
1101	$= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$ $= 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$ $= 8 + 4 + 0 + 1$ $= 13$

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

PARTICIPATION ACTIVITY 3.10.1: Binary number tool.

Set each binary digit for the unsigned binary number below to 1 or 0 to obtain the decimal equivalents of 9, then 50, then 212, then 255. Note that 255 is the largest integer that the 8 bits can represent.

0	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0
(decimal value)

PARTICIPATION ACTIVITY 3.10.2: Binary numbers.

1) Convert the binary number 00001111 to a decimal number.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

2) Convert the binary number 10001000 to a decimal number.

Check Show answer

3) Convert the decimal number 17 to an 8-bit binary number.

Check Show answer

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

4) Convert the decimal number 51 to an 8-bit binary number.

Check Show answer

CHALLENGE ACTIVITY 3.10.1: Create a binary number.

566436.4601014.qx3zqy7

Start

Convert the **decimal number** to binary before the **decimal number** reaches the binary numbers.

7

0 0 0 0

1 2 3

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Check Next

3.11 Additional practice: Health data

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.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

The following calculates a user's age in days based on the user's age in years.

zyDE 3.11.1: Health data: Age in days.

Load default template...

```
user_age_years = int(input('E

user_age_days = user_age_year

print(f'You are at least {use

```

22

Run

Create a different version of the program that:

1. Calculates the user's age in minutes and seconds.
2. Estimates the approximate number of times the user's heart has beaten in his/her lifetime using an average heart rate of 72 beats per minute.
3. Estimates the number of times the person has sneezed in his/her lifetime.
4. Estimates the number of calories that the person has expended in his/her lifetime (research on the internet to obtain a daily estimate). Also calculate the number of sandwiches (or other common food item) that equals that number of calories.
5. Be creative: Pick several interesting health-related statistics. Try searching the internet to determine how to calculate that data, and create a program to perform that calculation. The program can ask the user to enter any information needed to perform the calculation.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main

3.12 LAB: List basics

Given the user inputs, complete a program that does the following tasks:

- Define a list, my_list, containing the user inputs: my_flower1, my_flower2, and my_flower3 in the same order.
- Define a list, your_list, containing the user inputs, your_flower1 and your_flower2, in the same order.
- Define a list, our_list, by concatenating my_list and your_list.
- Append the user input, their_flower, to the end of our_list.
- Replace my_flower2 in our_list with their_flower.
- Remove the first occurrence of their_flower from our_list without using index().
- Remove the second element of our_list.

©zyBooks 11/21/24 13:10 2069507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Observe the output of each print statement carefully to understand what was done by each task of the program.

Ex: If the input is:

```
rose
peony
lily
rose
daisy
aster
```

the output is:

```
['rose', 'peony', 'lily', 'rose', 'daisy']
['rose', 'peony', 'lily', 'rose', 'daisy', 'aster']
['rose', 'aster', 'lily', 'rose', 'daisy', 'aster']
['rose', 'lily', 'rose', 'daisy', 'aster']
['rose', 'rose', 'daisy', 'aster']
```

566436.4601014.qx3zqy7

LAB ACTIVITY 3.12.1: LAB: List basics 10 / 10

©zyBooks 11/21/24 13:10 2069507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

main.py

Load default template...

```
my_flower1 = input()
my_flower2 = input()
my_flower3 = input()

your_flower1 = input()
your_flower2 = input()
```

```
your_flower2 = input()

their_flower = input()

# 1. TODO: Define my_list containing my_flower1, my_flower2, and my_flower3
# in that order
my_list = [my_flower1, my_flower2, my_flower3]

# 2. TODO: Define your_list containing your_flower1 and your_flower2
# in that order
your_list = [your_flower1, your_flower2]

# 3. TODO: Define our_list by concatenating my_list and your_list
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

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

Program output displayed here

Coding trail of your work What is this?

9/23 M0,10 min:10

3.13 LAB: Set basics

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Given the user inputs, complete a program that does the following tasks:

- Define a set, fruits, containing the user inputs: my_fruit1, my_fruit2, and my_fruit3.
- Add the user inputs, your_fruit1 and your_fruit2, to fruits.
- Add the user input, their_fruit, to fruits.
- Add your_fruit1 to fruits.
- Remove my_fruit1 from fruits.

Observe the output of each print statement carefully to understand what was done by each task of the program.

Note: For testing purposes, sets are printed using sorted() for comparison, as in the book's examples.

Ex: If the input is:

```
apple
peach
lemon
apple
pear
plum
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

the output is:

```
['apple', 'lemon', 'peach']
['apple', 'lemon', 'peach', 'pear']
['apple', 'lemon', 'peach', 'pear', 'plum']
['apple', 'lemon', 'peach', 'pear', 'plum']
['lemon', 'peach', 'pear', 'plum']
```

566436.4601014.qx3zqy7

LAB ACTIVITY | 3.13.1: LAB: Set basics | 10 / 10

main.py

Load default template...

```
my_fruit1 = input()
my_fruit2 = input()
my_fruit3 = input()

your_fruit1 = input()
your_fruit2 = input()

their_fruit = input()

# 1. TODO: Define a set, fruits, containing my_fruit1, my_fruit2, and my_fruit3
fruits = {my_fruit1, my_fruit2, my_fruit3}
print(sorted(fruits))

# 2. TODO: Add your_fruit1 and your_fruit2 to fruits
fruits.add(your_fruit1)
fruits.add(your_fruit2)
print(sorted(fruits))

```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

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.

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Run program

Input (from above) → main.py (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
9/23 M0,0,10 min:5
```

3.14 LAB: Input and formatted output: Right-facing arrow

Given input characters for an arrowhead and arrow body, print a right-facing arrow.

Ex: If the input is:

```
*
#
```

Then the output is:

```
      #
******##
******###
******##
      #
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

566436.4601014.qx3zqy7

LAB ACTIVITY 3.14.1: LAB: Input and formatted output: Right-facing arrow 10 / 10

main.py

Load default template...

```
base_char = input()
head_char = input()

row1 = '      ' + head_char
row2 = (base_char * 6) + (head_char * 2)
row3 = (base_char * 6) + (head_char * 3)



print(row1)
print(row2)
print(row3)
print(row2)
print(row1)
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

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

Program output displayed here

Coding trail of your work What is this?

9/23 M10 min:4

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

3.15 LAB: Phone number breakdown

Given an integer representing a 10-digit phone number, output the area code, prefix, and line number using the format (800) 555-1212.

Ex: If the input is:

```
8005551212
```

the output is:

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

```
(800) 555-1212
```

Hint: Use % to get the desired rightmost digits. Ex: The rightmost 2 digits of 572 is gotten by 572 % 100, which is 72.

Hint: Use // to shift right by the desired amount. Ex: Shifting 572 right by 2 digits is done by 572 // 100, which yields 5. (Recall integer division discards the fraction).

For simplicity, assume any part starts with a non-zero digit. So 0119998888 is not allowed.

566436.4601014.qx3zqy7

LAB ACTIVITY | 3.15.1: LAB: Phone number breakdown | 10 / 10

main.py

Load default template...

```
phone_number = int(input())

area_code = phone_number // 10000000
prefix3 = phone_number / 10000
prefix1 = (int(prefix3)) % (1000)
line_num = (phone_number - (prefix1 * 10000)) % 10000000

print(f'({area_code}) {prefix1}-{line_num}')
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

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

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Program output displayed here

Coding trail of your work What is this?

```
9/23 M5,10 min:7
```

3.16 LAB: Simple statistics

Given 4 floating-point numbers. Use a string formatting expression with conversion specifiers to output their product and their average as integers (rounded), then as floating-point numbers.

Output each rounded integer using the following:
`print(f'{your_value:.0f}')`

Output each floating-point value with three digits after the decimal point, which can be achieved as follows:
`print(f'{your_value:.3f}')`

Ex: If the input is:

```
8.3
10.4
5.0
4.8
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

the output is:

```
2072 7
2071.680 7.125
```

566436.4601014.qx3zqy7

LAB ACTIVITY

3.16.1: LAB: Simple statistics 10 / 10

main.py Load default template...

```
num1 = float(input())
num2 = float(input())
num3 = float(input())
num4 = float(input())

print(f'{num1*num2*num3*num4:.0f} {(num1+num2+num3+num4)/4:.0f}')


print(f'{num1*num2*num3*num4:.3f} {(num1+num2+num3+num4)/4:.3f}')
```

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

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

Program output displayed here

©zyBooks 11/21/24 13:10 2300507
Liz Vokac Main
KVCCCIS216JohnsonFall2024

Coding trail of your work What is this?

9/24 T0,10 min:7