21.1 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".

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 21.1.1: 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 21.1.1: A program with strings.

Try the 'mad libs' style game below.

Load default template...

```
#A 'Mad Libs' style game where user enters
#verbs, etc., and then a story using those

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

food = input('Enter a type of food: ')
print()

adjective = input('Enter an adjective: ')
print()

period = input('Enter a time period: ')
print()

# Tell the story
```

```
brother
burritos
macho
```

Run

PARTICIPATION ACTIVITY 21.1.2: 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

PARTICIPATION ACTIVITY 21.1.3: 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?

- `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 =

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

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.
```

PARTICIPATION ACTIVITY 21.1.4: 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:

Figure 21.1.2: Accessing individual characters of a string.

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

```
A B
Z
```

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

zyDE 21.1.2: String indexing.

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

Load default ter


```
1 alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2
3 user_number = int(input('Enter number to use as index: '))
4 print()
5
6 print('\nThe letter at index', user_number, 'of the alphabet is', alphabet[user
7
8
```

2

Run

PARTICIPATION ACTIVITY 21.1.5: 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.

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.

©zyBooks 12/15/22 00:42 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

Figure 21.1.3: Strings are immutable and cannot be changed.

Individual characters of a string cannot be directly changed.


```
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
```

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 21.1.4: String concatenation.

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

```
Easy as
abc123
```


PARTICIPATION ACTIVITY 21.1.6: String variables.

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

   - True
   - False

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 21.1.1: String basics.

422102.2723990.qx3zqy7

Start

Type the program's output

```
print('Tifa')
```

Tifa

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 21.1.2: Reading multiple data types.

Type two statements. The first reads user input into person_name. The second reads user input into person_age. Use the int() function to convert person_age into an integer. Below is a sample output for the given program if the user's input is: Amy 4

```
In 5 years Amy will be 9
```

Note: Do not write a prompt for the input values, use the format: variable_name = input()

422102.2723990.qx3zqy7


```
1
2 ''' Your solution goes here '''
3
4 print('In 5 years', person_name, 'will be', person_age + 5)
```

Run

CHALLENGE ACTIVITY 21.1.3: Concatenating strings.

Write two statements to read in values for my_city followed by my_state. Do not provide a prompt. Assign log_entry with current_time, my_city, and my_state. Values should be separated by a space. Sample output for given program if my_city is Houston and my_state is Texas:

```
2020-07-26 02:12:18: Houston Texas
```

Note: Do not write a prompt for the input values.

422102.2723990.qx3zqy7

```
current_time = '2020-07-26 02:12:18:'

''' Your solution goes here '''

print(log_entry)
```

Run

21.2 Additional practice: Grade calculation

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

exams.

zyDE 21.2.1: Grade calculator: Average score on three exams.

Load default ter


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

overall_grade = (exam1_grade + exam2_grade + exam3_grade) / 3

print('Your overall grade is:', overall_grade)

```

```
70
75
91
```

Run

Create a different version of the program that:

1. Calculates the overall grade for four equally weighted programming assignments, where 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, where 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) that account for 60% of the overall grade and four equally weighted programming assignments (graded out of 50) that account for 40% of the overall grade. Hint: The overall grade can be calculated as 0.6 * averageExamScore + 0.4 * averageProgScore.

21.3 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 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 int 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.

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

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

PARTICIPATION ACTIVITY 21.3.1: 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

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 below conversion functions:

Table 21.3.1: Conversion functions for some common types.

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

Converting a float to an int 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 21.3.1: Simple example of converting float and int types.

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

Load default template...

```
1 input_text = input('Enter a number:\n')
2 float_variable = float(input_text)
3 int_variable = int(float_variable)
4
5 print('original input text:', input_text)
6 print('input text converted to a float:', f
7 print('float variable converted to an int:'
8
```

18

Run

PARTICIPATION ACTIVITY 21.3.2: Type conversions.

What is the result of each expression?

1) int(1.55)

- 1.55
- 1
- '1.55'

2) float("7.99")

- 7.0
- 8.0
- 7.99

3) str(99)

CHALLENGE ACTIVITY 21.3.1: Type conversions.

422102.2723990.qx3zqy7

Start

Type the program's output

```
number = 1
new_number = number * 4
print(new_number)
```

4

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 21.3.2: Type conversion: Computing average owls per zoo.

Assign avg_owls with the average owls per zoo. The program will convert avg_owls to an integer and print the value.

Sample output for inputs: 1 2 4

```
Average owls per zoo: 2
```

422102.2723990.qx3zqy7

```
1 avg_owls = 0.0
2
3 num_owls_zooA = int(input())
4 num_owls_zooB = int(input())
5 num_owls_zooC = int(input())
6
7 ''' Your solution goes here '''
8
9 print(f'Average owls per zoo: {int(avg_owls)}')
```

Run

CHALLENGE ACTIVITY 21.3.3: Type conversion: Reading and adding values.

Assign total_owls with the sum of num_owls_A and num_owls_B. Remember num_owls_A and num_owls_B need to be converted to integers.

Sample output with inputs: 3 4

```
Number of owls: 7
```

422102.2723990.qx3zqy7

```
1 total_owls = 0
2
3 num_owls_A = input()
4 num_owls_B = input()
5
6 ''' Your solution goes here '''
7
8 print(f'Number of owls: {total_owls}')
```

Run

21.4 String formatting

Formatted string literals (f-strings)

Program output commonly includes expressions, like variables or other calculations, as a 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 a `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.

PARTICIPATION ACTIVITY 21.4.1: Creating literal strings with embedded expressions.

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 number amount.

PARTICIPATION ACTIVITY 21.4.2: 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')`

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 can be 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 **}}** can be used to place an actual curly brace into an f-string. Ex: `f'{{Jeff Bezos}}: Amazon'` produces the string "{Jeff Bezos}: Amazon".

Table 21.4.1: f-string examples.

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}`

PARTICIPATION ACTIVITY 21.4.3: 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

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

Check Show answer

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

Check Show answer

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), 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 a part of a format specification that determines how to represent a value in text form, such as integer (4), floating point (4.0), fixed precision decimal (4.000), percentage (4%), 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.

©zyBooks 12/15/22 00:42 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

Table 21.4.2: 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`
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 (6 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`
0[precision]d	Leading 0 notation	`number = 4` `print(f'{number:03d}')`	`004`

PARTICIPATION ACTIVITY 21.4.4: 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

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

CHALLENGE ACTIVITY 21.4.1: String formatting with f-strings.

422102.2723990.qx3zqy7

Start

Type the program's output

```
name = 'Sue'

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

Your name is Sue

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 21.4.2: Printing an f-string.

Write a *single* statement to print: user_word,user_number. Note that there is no space between the comma and user_number.

Sample output with inputs: Amy 5

```
Amy,5
```


422102.2723990.qx3zqy7

```
1 user_word = input()
2 user_number = int(input())
3
4 ''' Your solution goes here '''
5
```

Run

CHALLENGE ACTIVITY 21.4.3: String formatting.

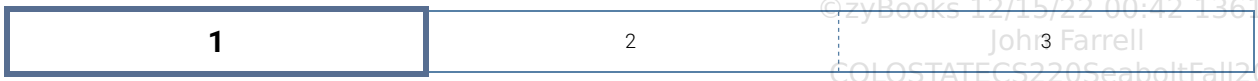
422102.2723990.qx3zqy7

Start

Select the most appropriate replacement field definitions.

num_students = 71

print(f'The math class has [lots of ▾] students.')



Check Next

21.5 Binary numbers

Binary numbers

Normally, a programmer can think in terms of base ten 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 very 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 a common and 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 21.5.1: Decimal numbers use weighed powers of 10.

Decimal number with 3 digits	Representation
212	$= 2 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0$ $= 2 \cdot 100 + 1 \cdot 10 + 2 \cdot 1$ $= 200 + 10 + 2$ $= 212$

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

Table 21.5.2: Binary numbers use weighed powers of 2.

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

PARTICIPATION ACTIVITY 21.5.1: 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 also that 255 is the largest integer that the 8 bits can represent.

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

0 (decimal value)

PARTICIPATION ACTIVITY 21.5.2: Binary numbers.

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

Check Show answer

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

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

Check Show answer

CHALLENGE ACTIVITY 21.5.1: Create a binary number.

422102.2723990.qx3zqy7

Start

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

5

0 0 0

0

©zyBooks 12/15/22 00:42 1361995
John Farrell
COLOSTATECS220SeaboltFall2022