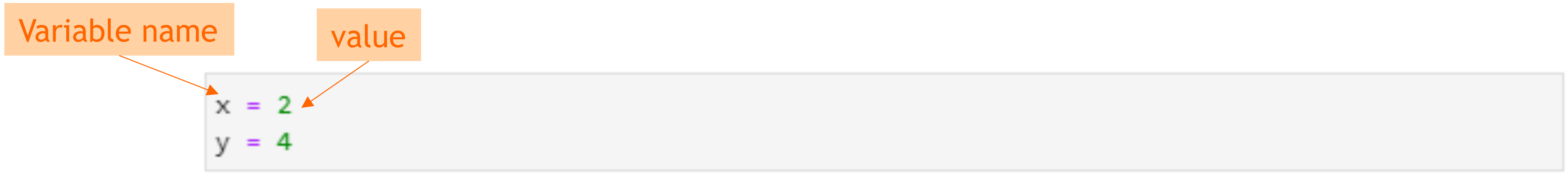
Variables and Data Types

Agenda

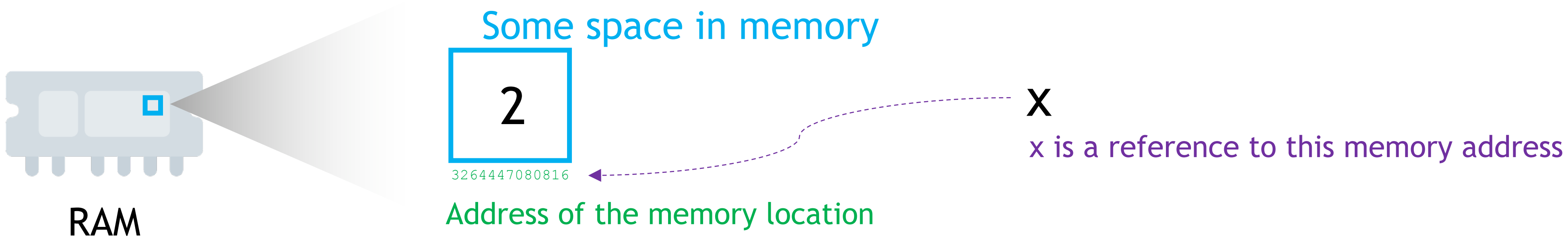
- Variables
- Data types
  - String
  - Number
- Input and Output
- Error message

Variables

- **Variables:**
  - Variables are used to access and manipulate data stored in memory.
  - A variable is created the moment you first assign a value to it.



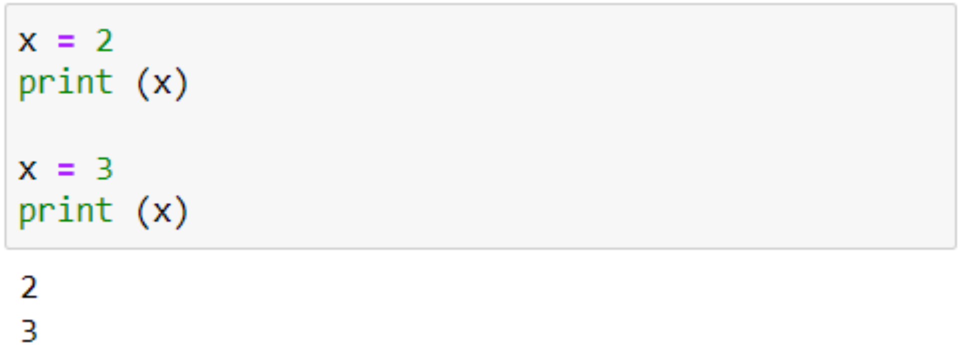
```
x = 2
y = 4
```


Some space in memory
2
3264447080816
Address of the memory location
x
x is a reference to this memory address
RAM

- RAM is short for "random access memory".
- RAM is the main memory of a computer system.

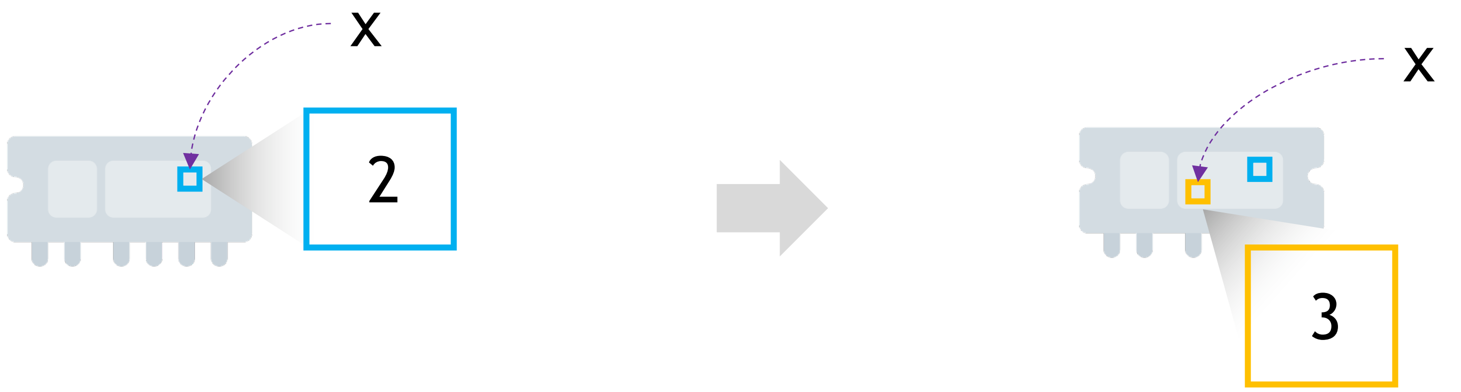
Variable reassignment

- Variable assignment is the process of assigning a new value to an existing variable.



```
x = 2
print (x)

x = 3
print (x)
```

```
2
3
```



At some point, Python's garbage collector will recognize that the data 2 is no longer used or referenced. It will then free the memory associated with the data 2 so it can be reused for other purposes.

Variable naming rules

- Rules for naming variables in Python:
  - Variable name cannot be a Python reserved word.
    - class = "A001" ✗
    - class_id = "A001" ✓
  - Variable name cannot contain spaces
    - Item price = 150 ✗
    - Item_price = 150 ✓
  - Variable names cannot begin with a digit (0-9). They must start with a letter (a-z or A-Z) or an underscore (_). Letters, numbers, or underscores can be used after the first character.
    - 1name = 30 ✗
    - name1 = 30 ✓
  - Variable names are case sensitive.
    - WORD = "first"
    - word = "second"

- Variable name should reflect its use.
  - Choose descriptive and meaningful names for variables that indicate the purpose of the variable.

Python reserved words

Reserved words in Python are also known as keywords. These are words that have a special meaning and functionality within the Python programming language. Therefore, they cannot be used as identifiers, such as variable names and function names.

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

```
help("keywords")
```

Data Type

Data types

Data types in programming are important because they allow us to represent and manipulate different kinds of data in a structured and efficient way.

Name	Type	Description	Example
String	str	A sequence of characters	“hello”, ‘course’, “covid-19”, “2”
Integer	Int	Whole numbers	2, 4, 100, 4000
Float	float	Numbers containing one or more decimals	3.8, 50.9, 100.0
Booleans	bool	Logical value indicating TRUE or FALSE	True, False
List	list	Ordered sequence of objects	[“hello”, “world”,”2021”] [“hello, 5, 100.0]
Dictionary	dict	Key: value pairs	{“key1”: name1, “key2”:name2}
Tuples	tup	Ordered immutable sequence of objects	(10,20) (“hello”, “world”)

Sets	set	Unordered collection of unique objects	{2,4,6,8}
			{3,”hello”, 50.9}

Types of values/variables

- Use `type` function to get the type of value

```
type("hello world")
```

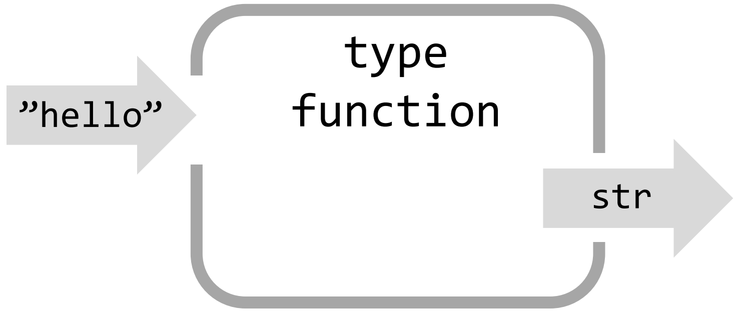
```
str
```

```
type(100.1)
```

```
float
```

```
type("100.1")
```

```
str
```



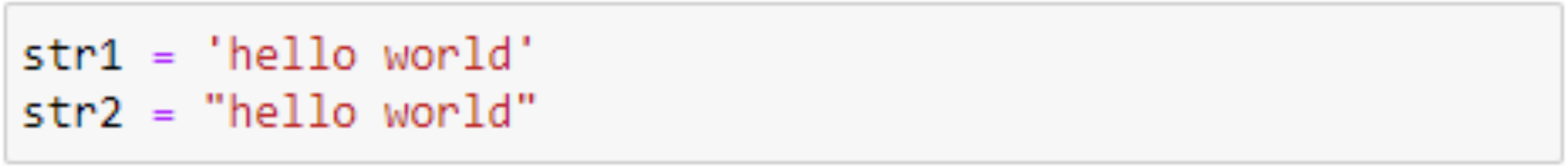
- Check the type of variable

```
x = 10
type(x)
```

- A function is a piece of ready-made code that can be reused.
- Python built-in functions allow you to perform common tasks without writing code from scratch.

String

- Must be enclosed in single (‘) or double (“) quote marks

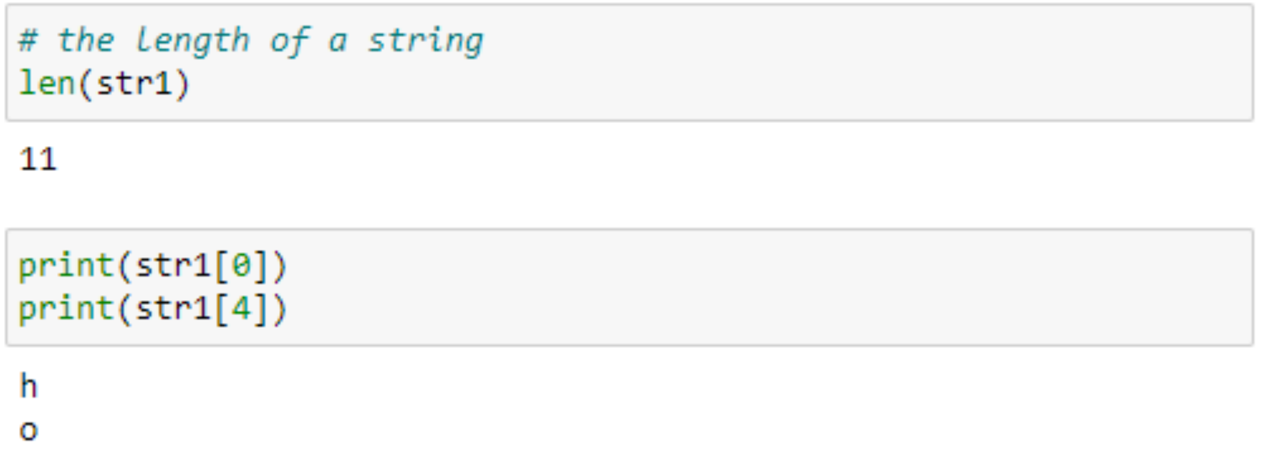


```python
str1 = 'hello world'
str2 = "hello world"
```

String – indexing

- A string can be thought of as a list of characters.
- An index refers to a position within an ordered list.
- Each character is given an index from zero (at the beginning) to the length minus one (at the end).

h	e	l	l	o		w	o	r	l	d	→ character
0	1	2	3	4	5	6	7	8	9	10	→ Index



```
# the length of a string
len(str1)
```

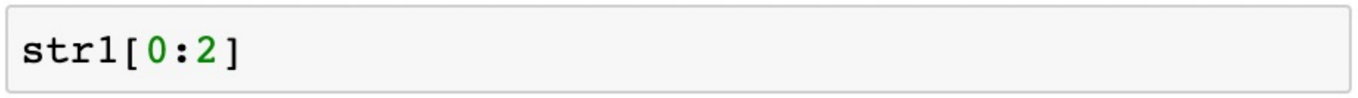
```
11
```

```
print(str1[0])
print(str1[4])
```

```
h
o
```

String – slicing

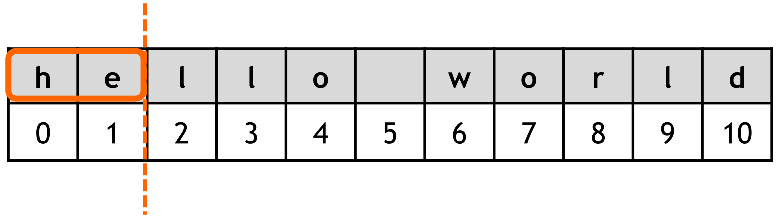
- Specify the start index and the end index, separated by a colon, to get a part of the string.
- Format: `[ start_index : stop_index ]`

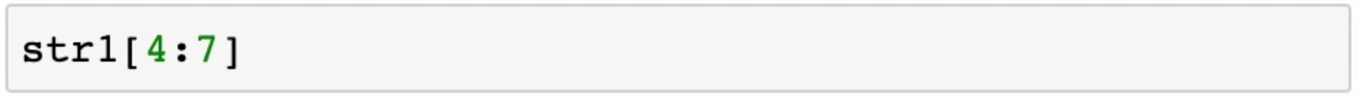


```
str1[0:2]
```

```
'he'
```

From position 0 to position 2, but NOT including 2.

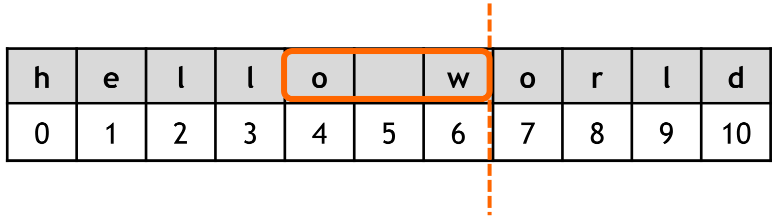




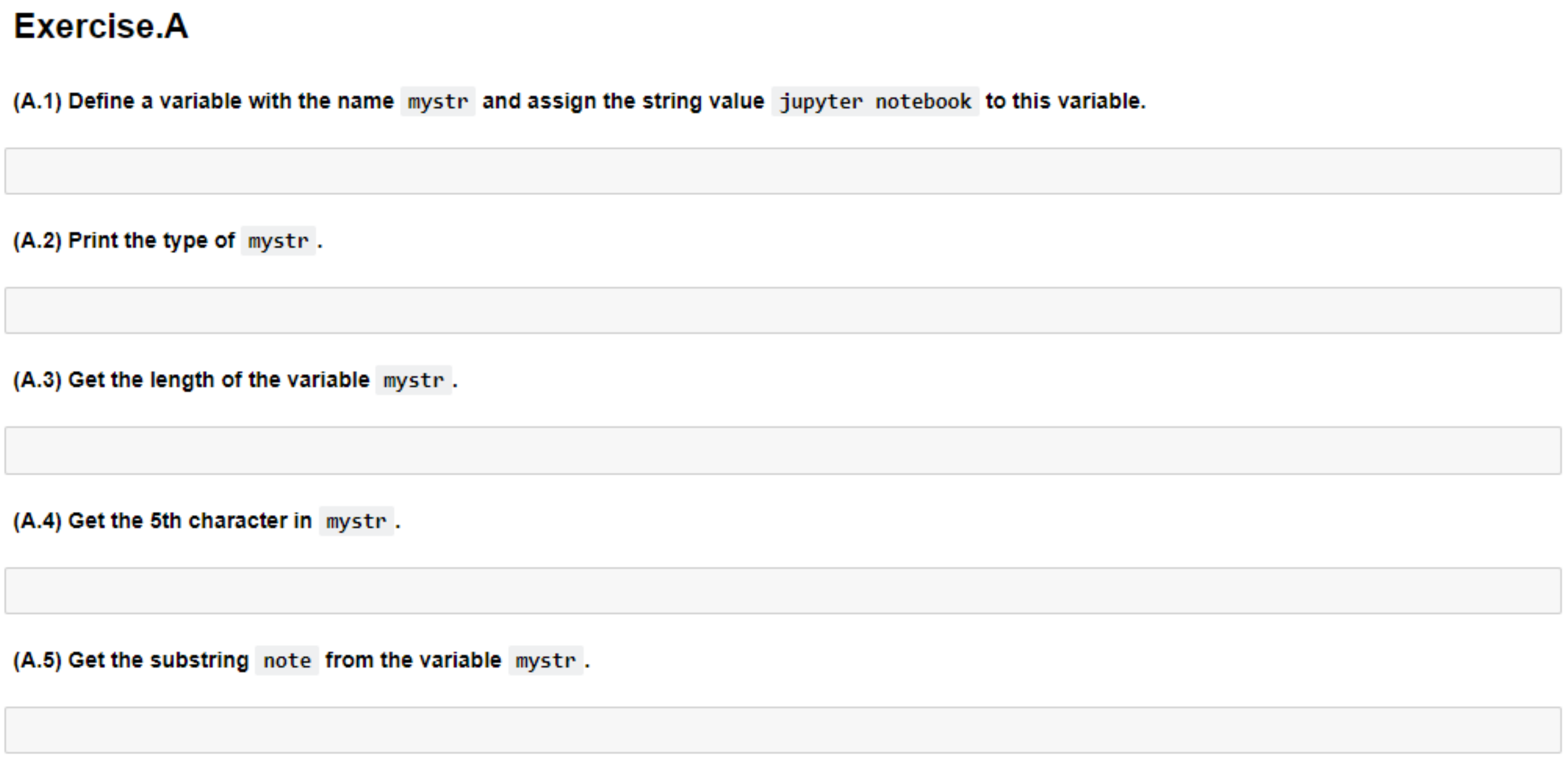
```
str1[4:7]
```

```
'o w'
```

From position 4 to position 7, but NOT including 7.



Exercise



Exercise.A

(A.1) Define a variable with the name `mystr` and assign the string value `jupyter notebook` to this variable.

(A.2) Print the type of `mystr`.

(A.3) Get the length of the variable `mystr`.

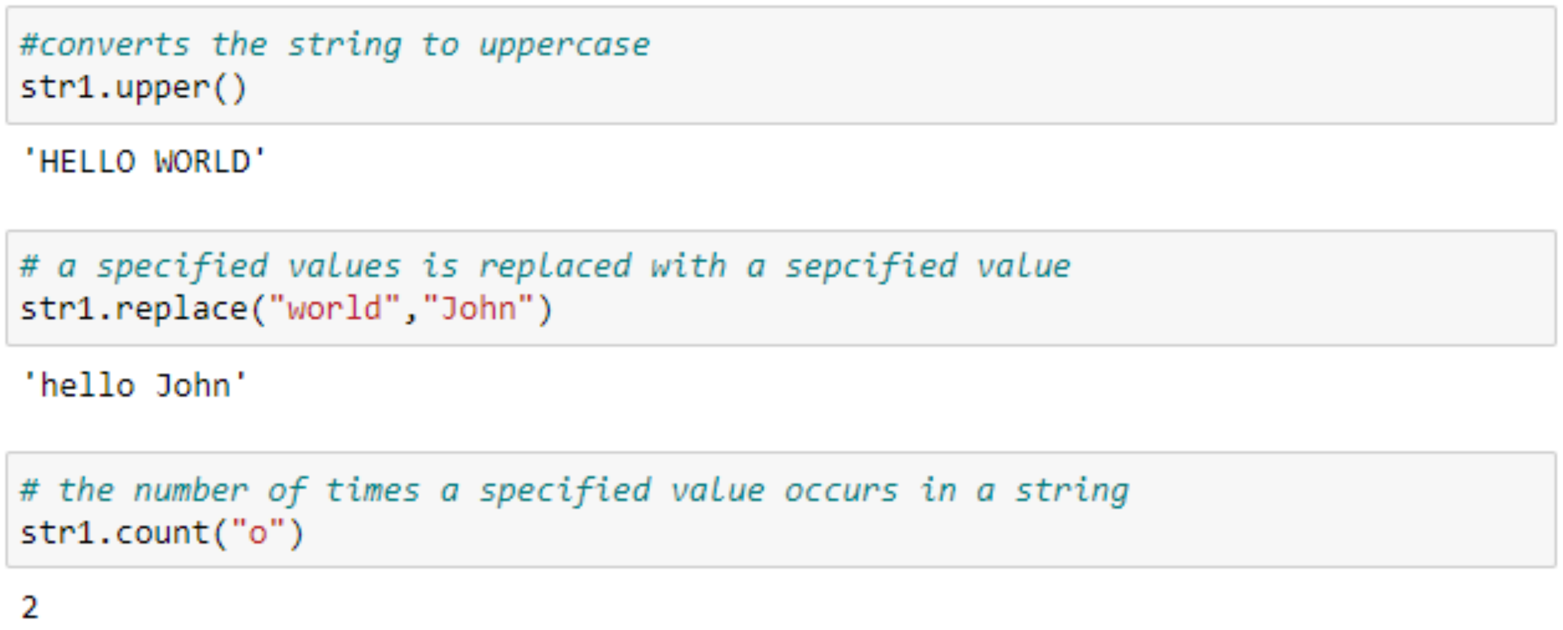
(A.4) Get the 5th character in `mystr`.

(A.5) Get the substring `note` from the variable `mystr`.

String - Methods

- Python has a set of built-in methods that you can use on strings

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10



```
#converts the string to uppercase
str1.upper()
```

```
'HELLO WORLD'
```

```
# a specified values is replaced with a sepcified value
str1.replace("world","John")
```

```
'hello John'
```

```
# the number of times a specified value occurs in a string
str1.count("o")
```

```
2
```

- Python method is like a function, but it is associated to certain data type.
- Strings are immutable, so you cannot change the contents of string variables.
- A possible solution is to create a new string variable with the necessary modifications.

String – Methods

- Use `find()` to find the first occurrence of the specified value.

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
# search for the position of the letter
str1.find("w")
```

```
6
```

```
# search for the position of the substring
str1.find("world")
```

```
6
```

- It returns `-1` if the value is not found.

```
str1.find("word")
```

```
-1
```

- Use dir(str) to list the methods available for string.

String operators

- Use the + operator to concatenate two strings.

```
s1 = "hello"
s2 = "world"
```

```
s1 + s2
```

```
'helloworld'
```

- Use the in operator to check if a string contains another string.

```
str1 = "hello world"
```

```
"world" in str1
```

```
True
```

```
"word" in str1
```

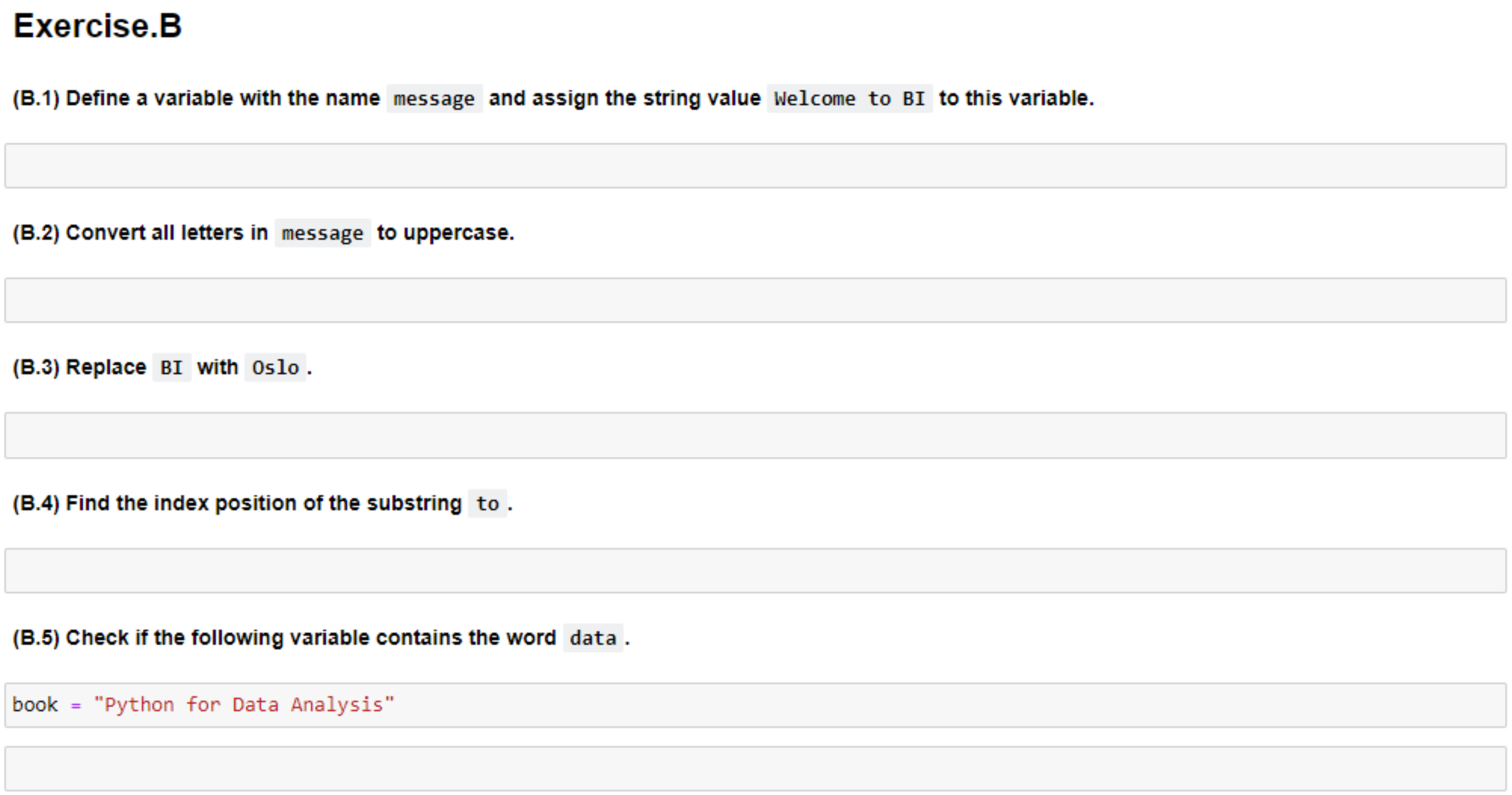
```
False
```

Operators

- Operators are symbols that perform operations on variables or values.

Type of operator	Examples
Arithmetic operators	+, -, *, /
Assignment operators	=, +=, -=
Comparison operators	==, !=, <, >
Logical operators	and, or, not
Membership operators	in, not in
Identity operators	is, is not
Bitwise operators	&, |, ^

Exercise



Exercise.B

(B.1) Define a variable with the name `message` and assign the string value `Welcome to BI` to this variable.

(B.2) Convert all letters in `message` to uppercase.

(B.3) Replace `BI` with `Oslo`.

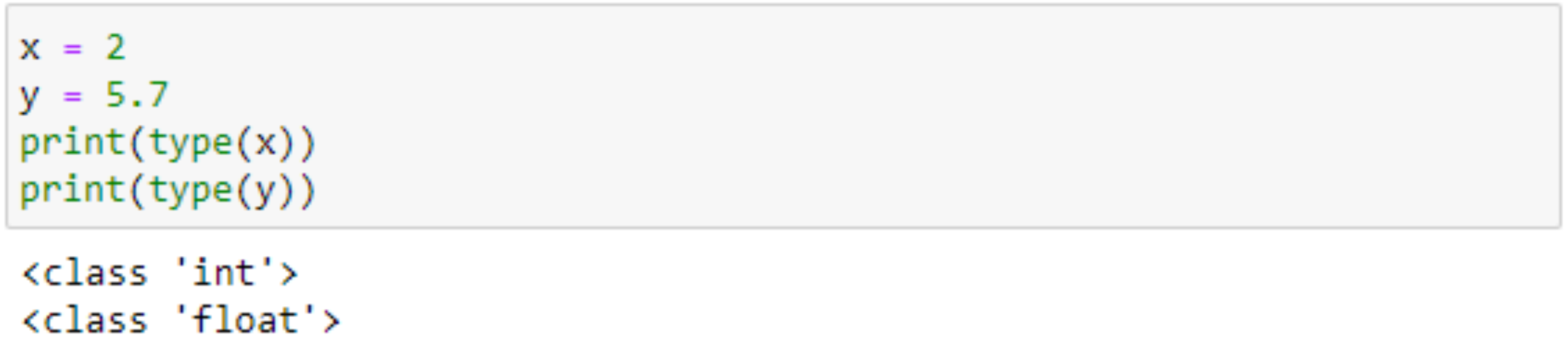
(B.4) Find the index position of the substring `to`.

(B.5) Check if the following variable contains the word `data`.

```
book = "Python for Data Analysis"
```

Numbers – integer and float

- Integer: Whole numbers
- Float: Numbers containing one or more decimals



```
x = 2
y = 5.7
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'float'>
```

Numbers – integer and float

- Functions for numbers

```
# returns the absolute value of the given number
x = -9
abs(x)
```

```
9
```

```
# round a number
y = 21.9267
print(round(y))
print(round(y,2))
```

(round to the nearest integer) 21.9267

(rounded to the second decimal place) 21.9267

```
22
21.93
```

- Type conversion

```
# float to integer
print(int(y))
```

(chop off the decimal portion of a number) 21.~~9267~~

```
21
```

```
# integer to flaot
print(float(x))
```

```
-9.0
```

BI

AACSB ACCREDITED EFMD EQUIS ACCREDITED ASSOCIATION OF MBAS AMBA ACCREDITED

Convert string to integer

- Convert string to integer and assign the result to a new variable.

```
z1 = "20"
print(type(z1))
```

```
<class 'str'>
```

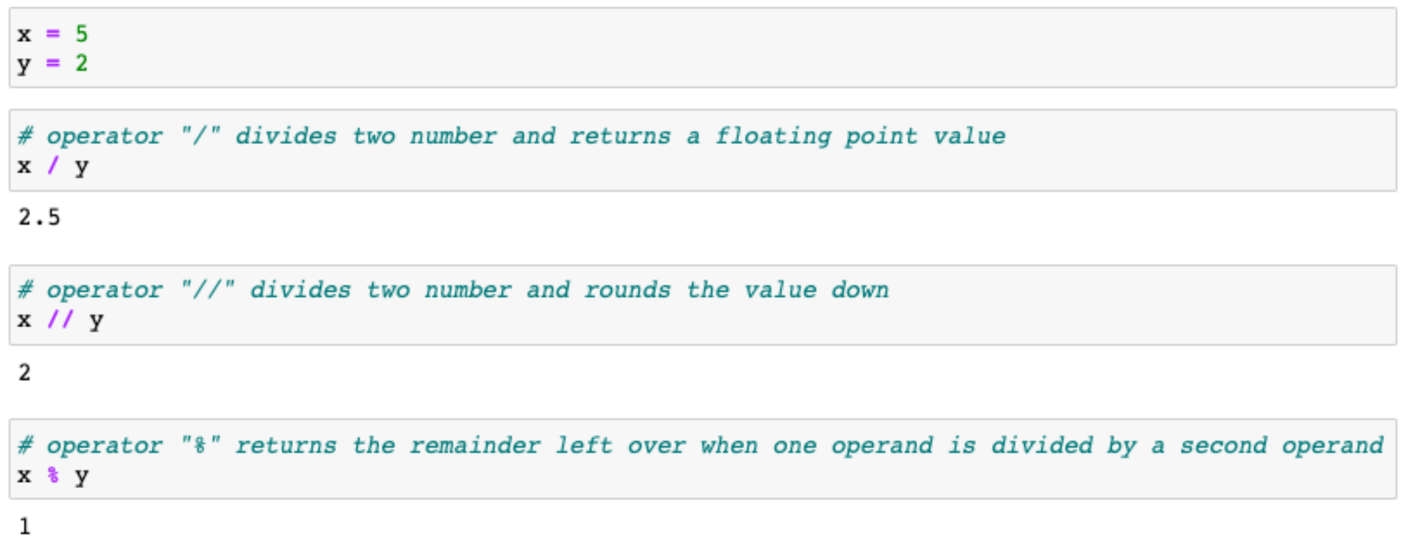
```
z2 = int(z1)
print(type(z2))
```

```
<class 'int'>
```

Arithmetic operators

- Arithmetic operators are used with numeric data types to perform common mathematical operations.
- Python follows the standard rules of mathematics to determine the order of operations.

Arithmetic operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
//	Floored division
%	Modulo



```
x = 5
y = 2
```

```
# operator "/" divides two number and returns a floating point value
x / y
```

```
2.5
```

```
# operator "//" divides two number and rounds the value down
x // y
```

```
2
```

```
# operator "%" returns the remainder left over when one operand is divided by a second operand
x % y
```

```
1
```

Parentheses ➔ Exponentiation ➔ Multiplication, Division, Floor Division, Modulo ➔ Addition, Subtraction

Exercise

Exercise.C

(C.1) Define a variable `n = 15.27391`. Print the data type of `n`.

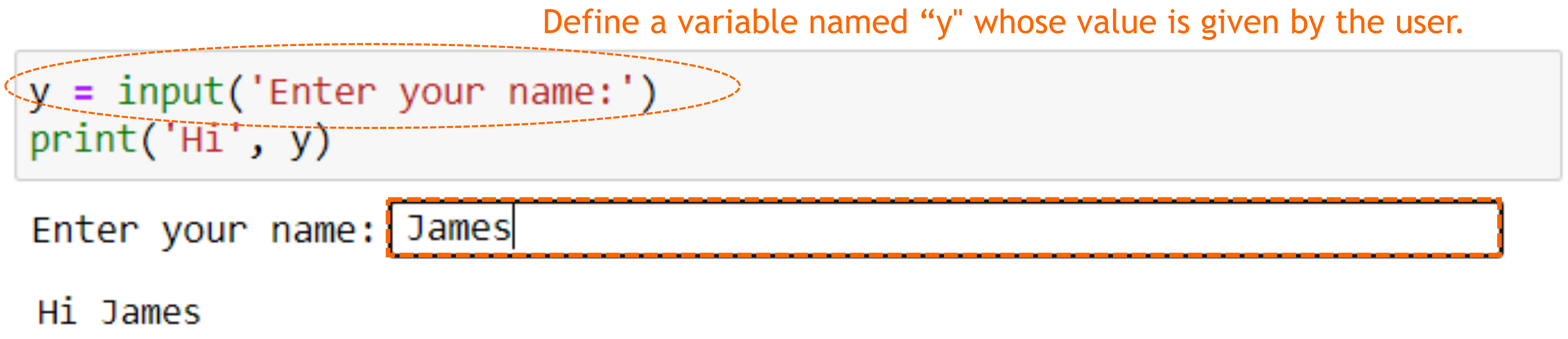
(C.2) Round `n` to two decimal places.

(C.3) Calculate $\frac{(n+2)}{3}$.

Input and Output

Input

- Most programs need to read input from the user. The simplest way to accomplish this in Python is with `input()` to create an input box.



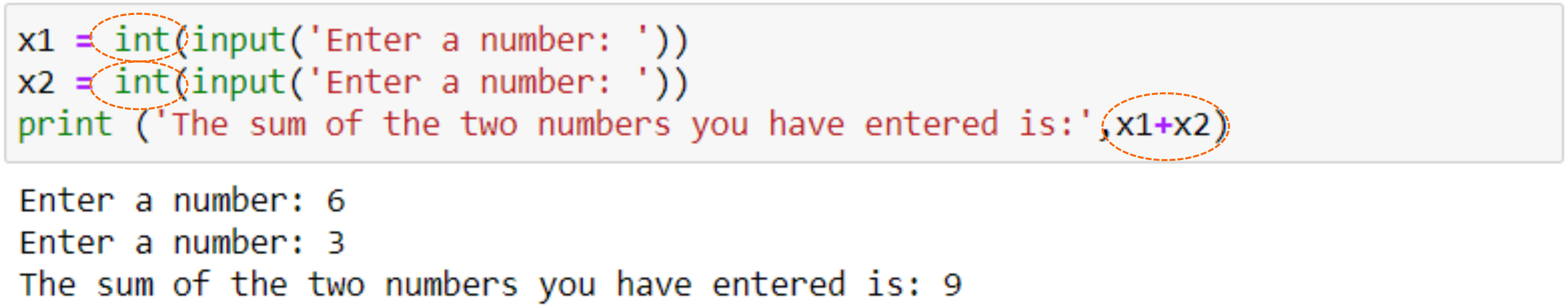
Define a variable named “y" whose value is given by the user.

```
y = input('Enter your name:')
print('Hi', y)
```

```
Enter your name: James

Hi James
```

Input

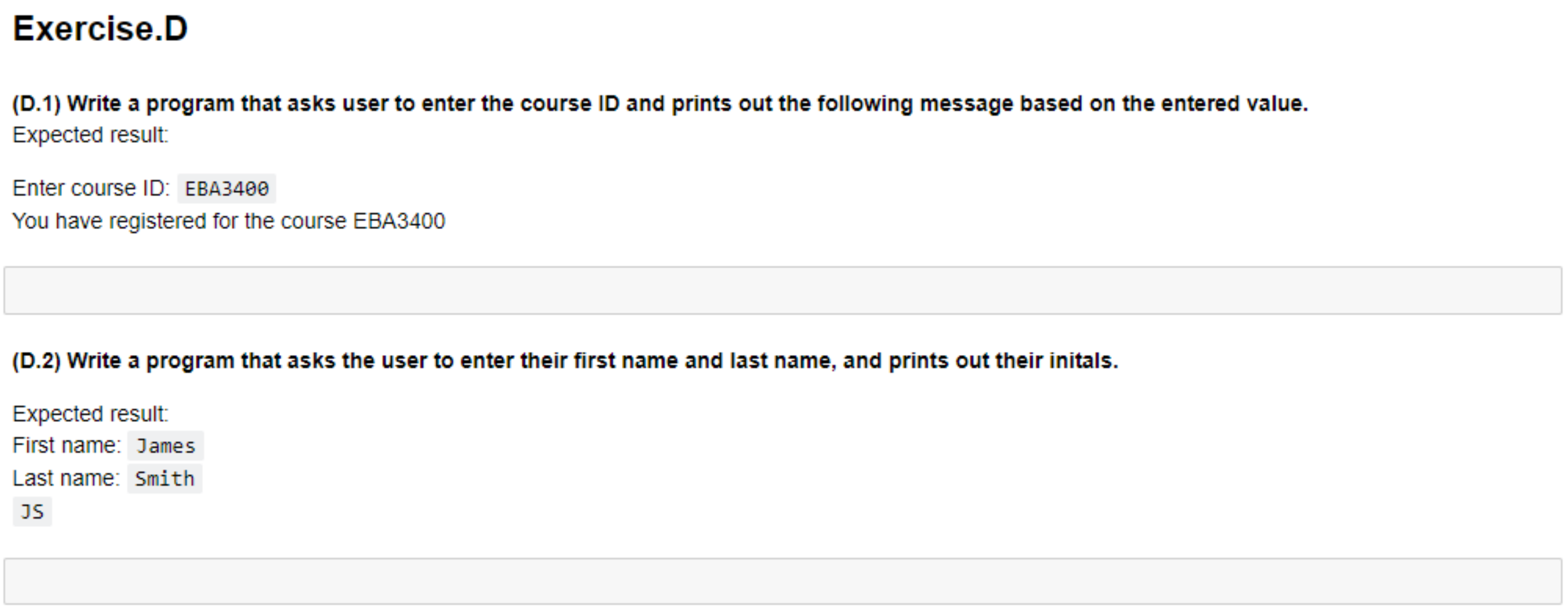
- input() always returns a `string`. If you want a numeric type, then you need to convert the string to the appropriate type.



```
x1 = int(input('Enter a number: '))
x2 = int(input('Enter a number: '))
print ('The sum of the two numbers you have entered is:',x1+x2)
```

```
Enter a number: 6
Enter a number: 3
The sum of the two numbers you have entered is: 9
```

Exercise



Exercise.D

(D.1) Write a program that asks user to enter the course ID and prints out the following message based on the entered value.

Expected result:

```
Enter course ID: EBA3400
You have registered for the course EBA3400
```

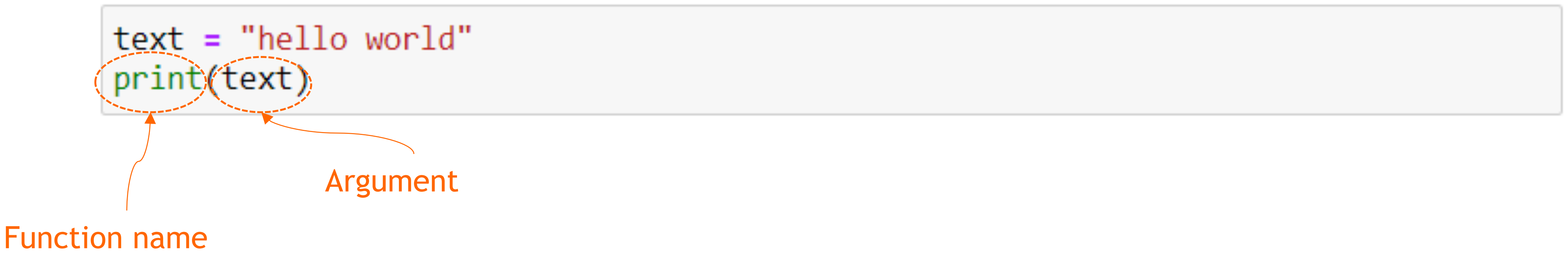
(D.2) Write a program that asks the user to enter their first name and last name, and prints out their initals.

Expected result:

```
First name: James
Last name: Smith
JS
```

Output – print function

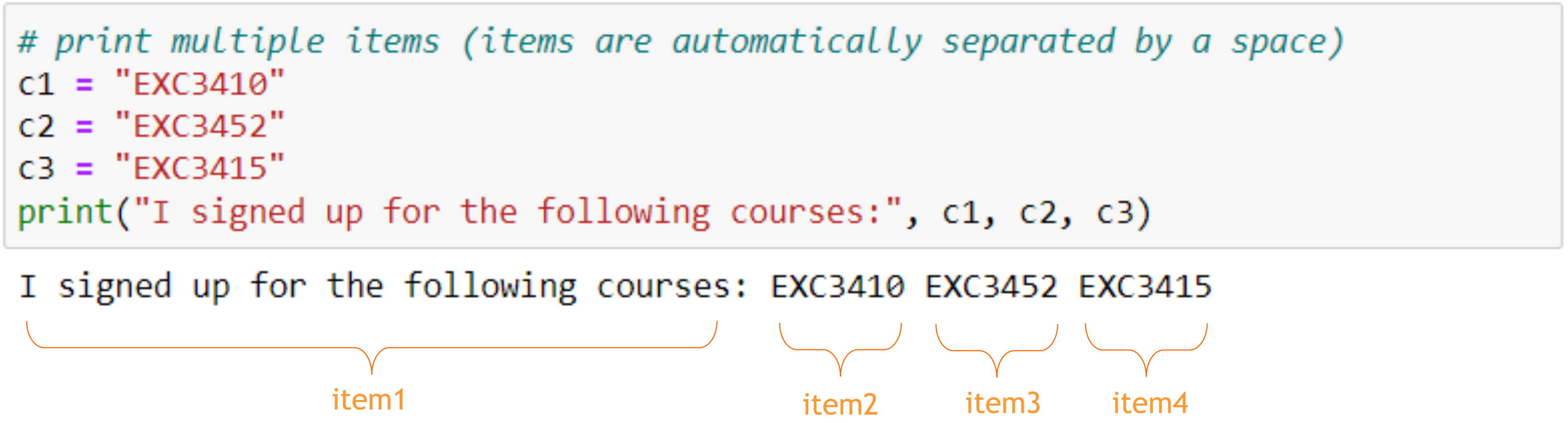
- Display output with `print` function
  - Function: piece of prewritten code that performs an operation.
  - Argument: data given to a function



```
text = "hello world"
print(text)
```

Output – print multiple items

- Displaying multiple items with the print function
  - Items are separated by commas when passed as arguments
  - Items are automatically separated by a space when displayed on screen



```python
# print multiple items (items are automatically separated by a space)
c1 = "EXC3410"
c2 = "EXC3452"
c3 = "EXC3415"
print("I signed up for the following courses:", c1, c2, c3)
```

```
I signed up for the following courses: EXC3410 EXC3452 EXC3415
```

https://realpython.com/python-string-formatting/

Output – optional arguments

- Use optional arguments to control the formatting.
  - ”sep" is used to specify the separator between multiple items, the default is a space (” “).

```python
print("A", "B", "C", sep = "-")
```

```
A-B-C
```

  - ”end" is used to specify what character should be added at the end, the default is a newline (”\n").

```python
print("A")
print("B")
print("C")
```

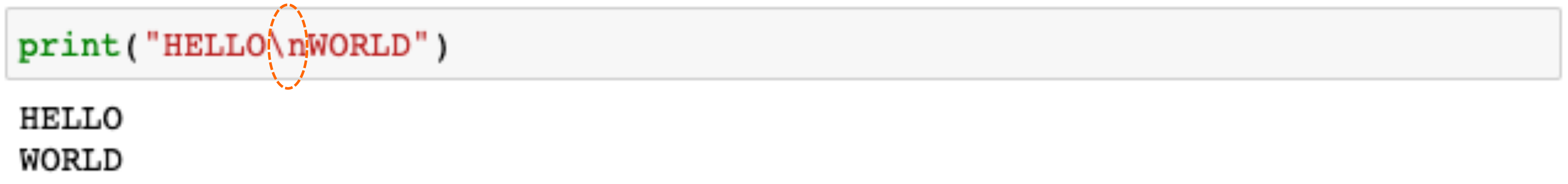
```
A
B
C
```

```python
print("A", end = "_")
print("B", end = "_")
print("C", end = "_")
```

```
A_B_C_
```

Output – print multiple lines

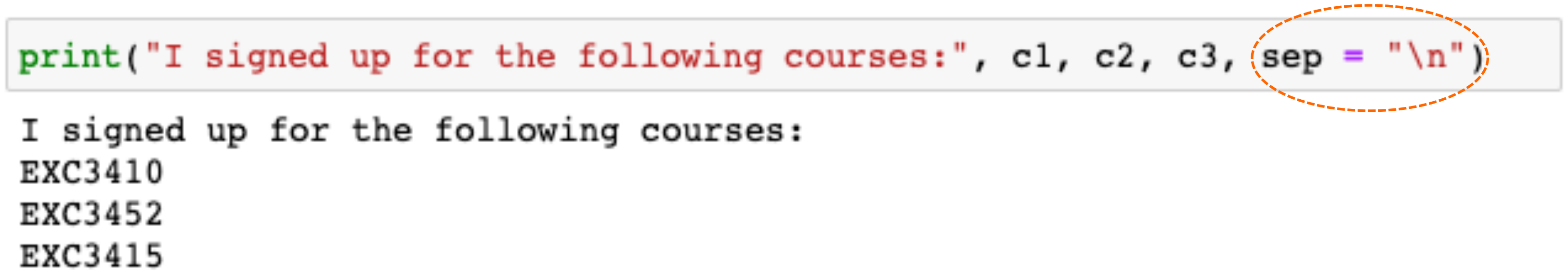
- Displaying multiple lines with the print function.
  - Split text into multiple lines by a new line code (\n).



```
print("HELLO\nWORLD")
```

```
HELLO
WORLD
```

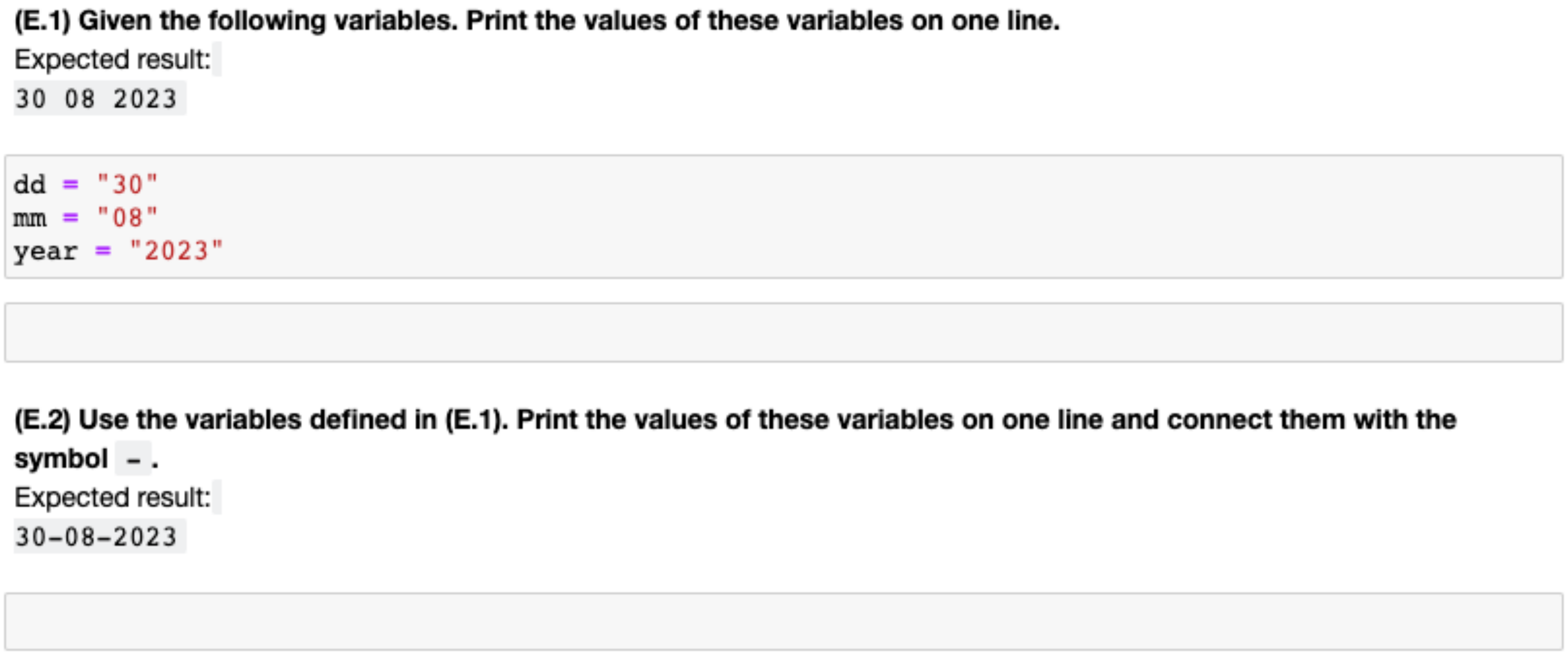
- Try to use “sep” to print multiple items by line.



```
print("I signed up for the following courses:", c1, c2, c3, sep = "\n")
```

```
I signed up for the following courses:
EXC3410
EXC3452
EXC3415
```

Exercise



(E.1) Given the following variables. Print the values of these variables on one line.

Expected result:

```
30 08 2023
```

```python
dd = "30"
mm = "08"
year = "2023"
```

(E.2) Use the variables defined in (E.1). Print the values of these variables on one line and connect them with the symbol `-`.

Expected result:

```
30-08-2023
```

Output - String formatting

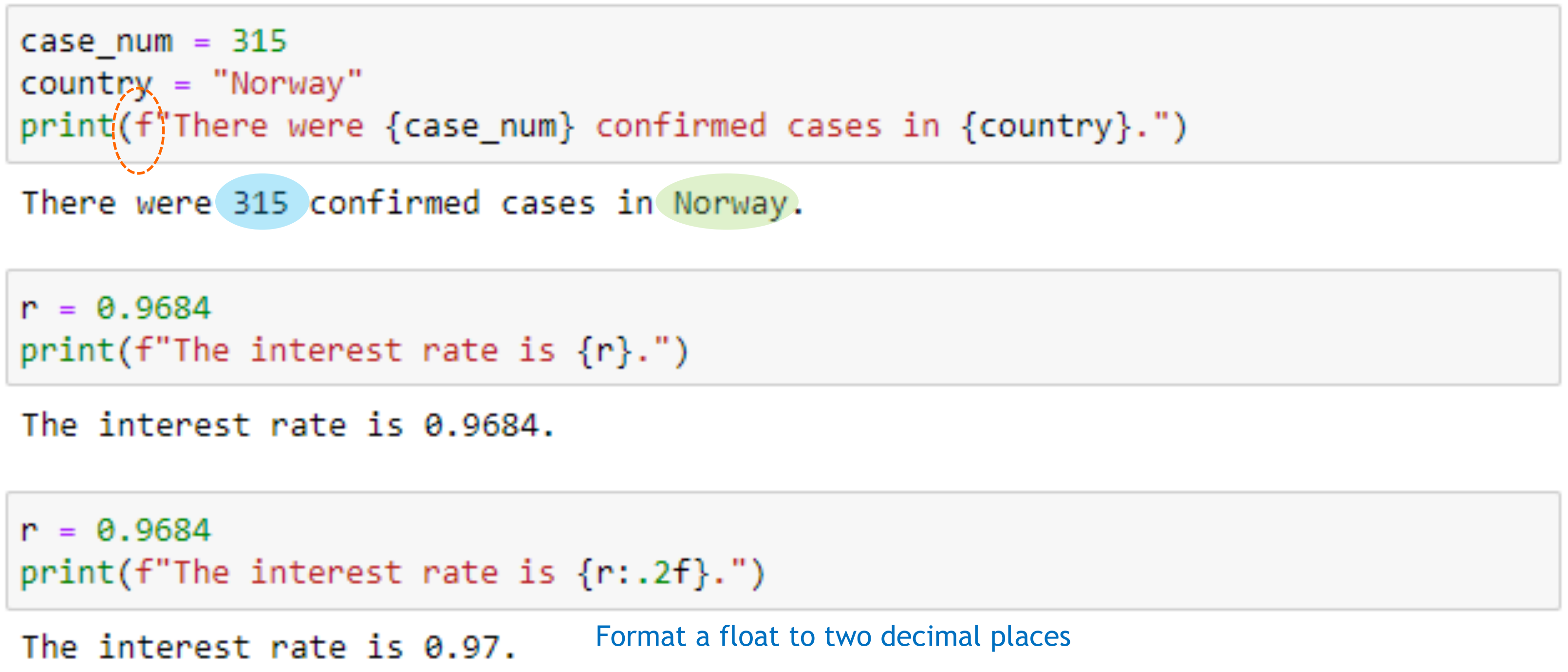
- String formatting is the process of inserting a <u>custom string</u> or <u>variable</u> in predefined text.

```
There were _____ confirmed cases in _______.

There were  225  confirmed cases in   Oslo  .
There were  160  confirmed cases in  Bergen .
```

- Different ways of string formatting in Python:
  1) `f-strings`
  2) `str.format()` (optional)
  3) `%` string formatter (optional)

Output – (1) f-strings

- Format a string by simply prefixing it with the letter "f“, and specify variables (or expression) in curly brackets.



```
case_num = 315
country = "Norway"
print(f"There were {case_num} confirmed cases in {country}.")
```

```
There were 315 confirmed cases in Norway.
```

```
r = 0.9684
print(f"The interest rate is {r}.")
```

```
The interest rate is 0.9684.
```

```
r = 0.9684
print(f"The interest rate is {r:.2f}.")
```

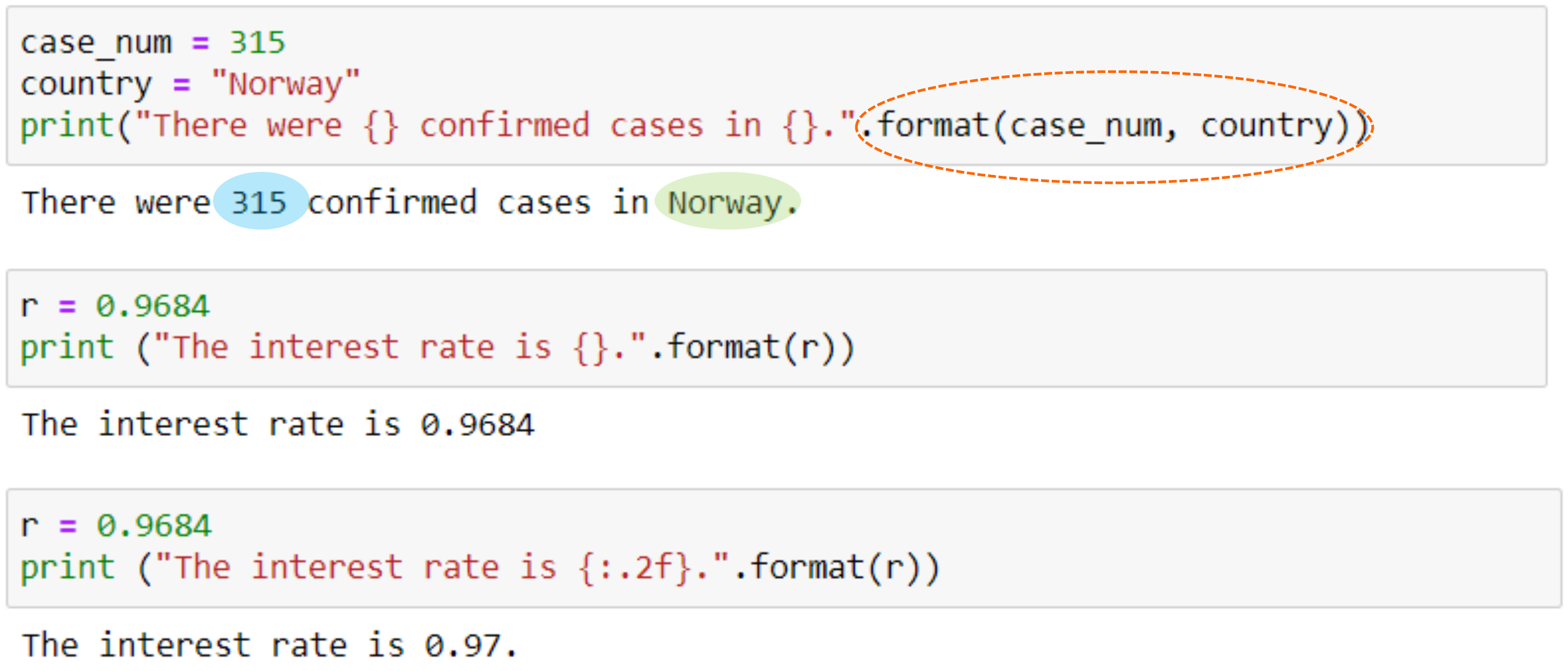
```
The interest rate is 0.97.
```

Format a float to two decimal places

- 'fstring' is short for 'formatted string’.
- ‘fstrings’ was introduced in Python 3.6

Output – (2) str.format()

- Create placeholders with curly brackets and then use the `format()` method to replace those placeholders with the provided values.



```
case_num = 315
country = "Norway"
print("There were {} confirmed cases in {}.".format(case_num, country))
```

```
There were 315 confirmed cases in Norway.
```

```
r = 0.9684
print ("The interest rate is {}.".format(r))
```

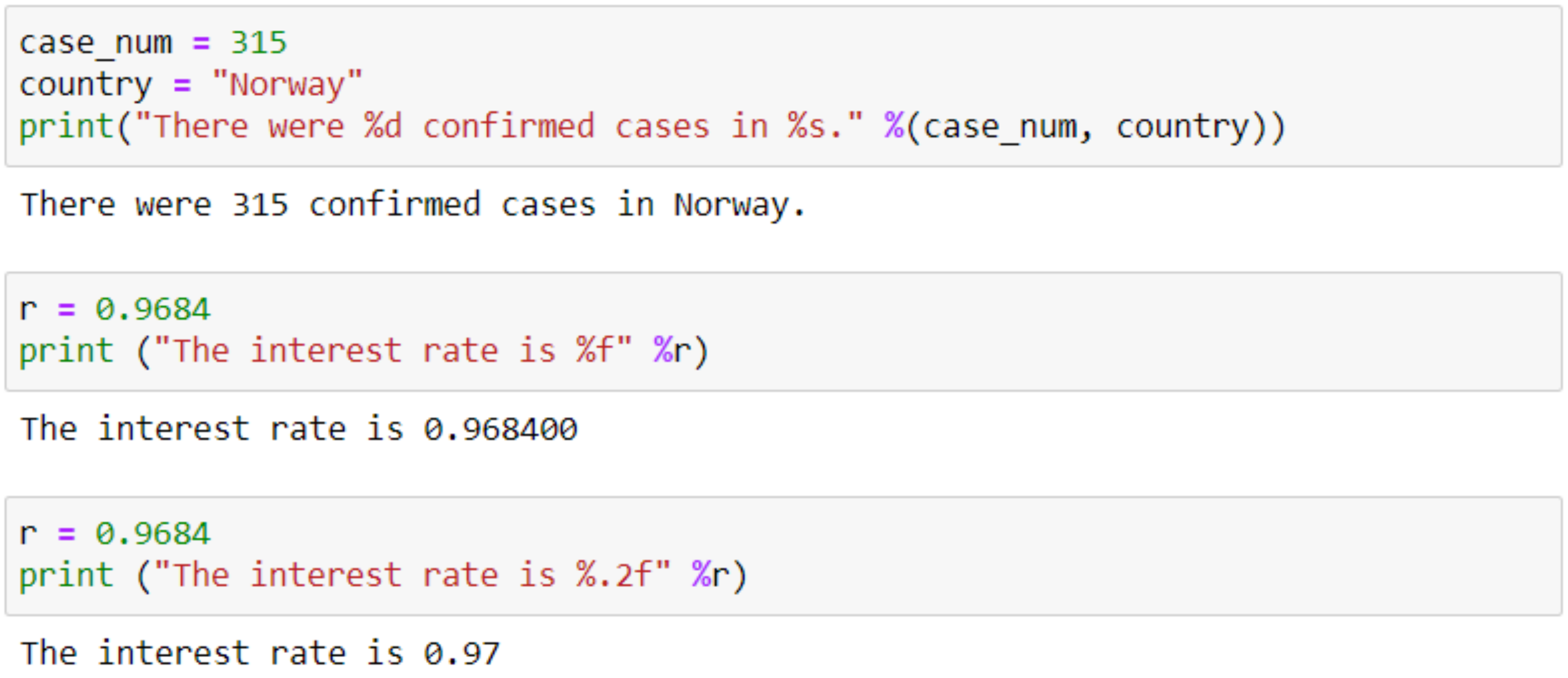
```
The interest rate is 0.9684
```

```
r = 0.9684
print ("The interest rate is {:.2f}.".format(r))
```

```
The interest rate is 0.97.
```

Output – (3) % str formatter

- The `%` operator can also be used for string formatting.
  - `%s` for strings, `%d` for integers, `%f` for floats, and `%r` for any value.



```
case_num = 315
country = "Norway"
print("There were %d confirmed cases in %s." %(case_num, country))
```

```
There were 315 confirmed cases in Norway.
```

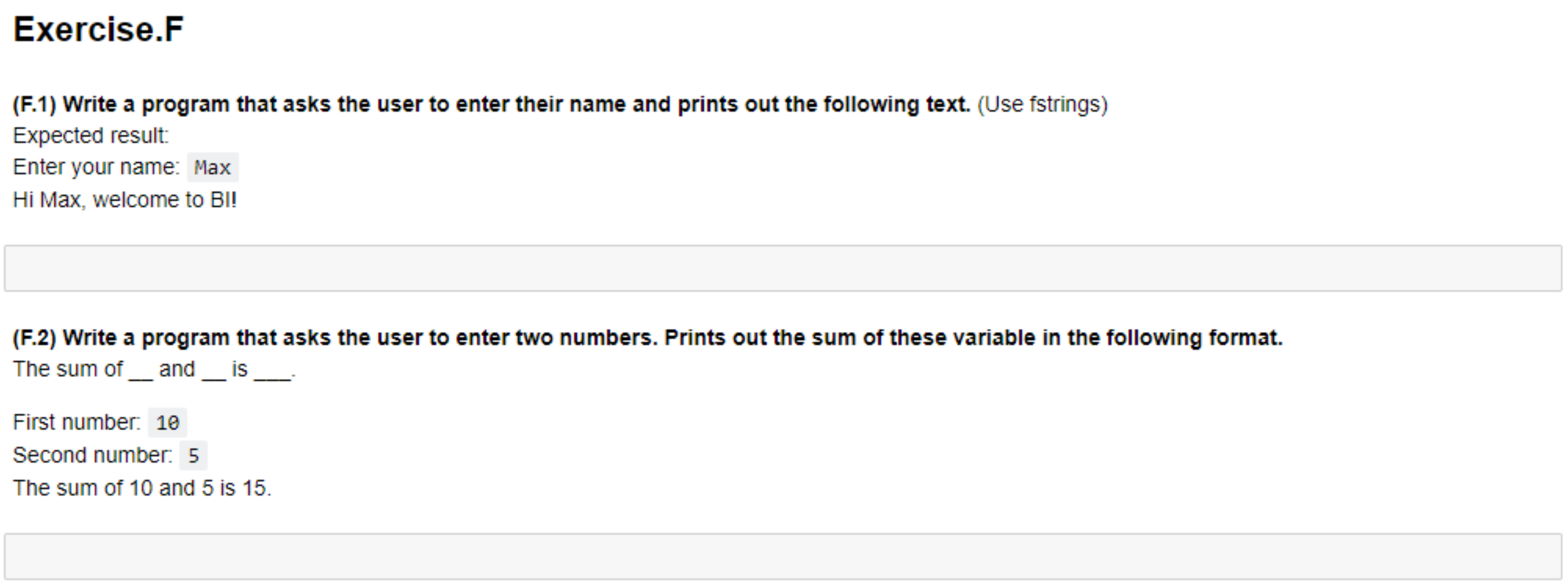
```
r = 0.9684
print ("The interest rate is %f" %r)
```

```
The interest rate is 0.968400
```

```
r = 0.9684
print ("The interest rate is %.2f" %r)
```

```
The interest rate is 0.97
```

Exercise



Exercise.F

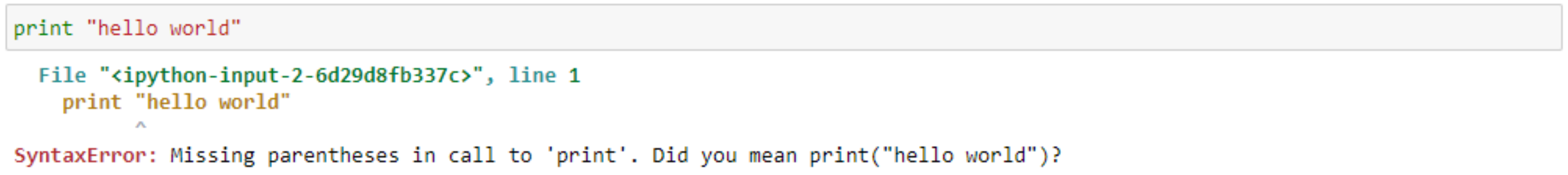
(F.1) Write a program that asks the user to enter their name and prints out the following text. (Use fstrings)
Expected result:
Enter your name: `Max`
Hi Max, welcome to BI!

(F.2) Write a program that asks the user to enter two numbers. Prints out the sum of these variable in the following format.
The sum of __ and __ is ___.

First number: `10`
Second number: `5`
The sum of 10 and 5 is 15.

Types of errors

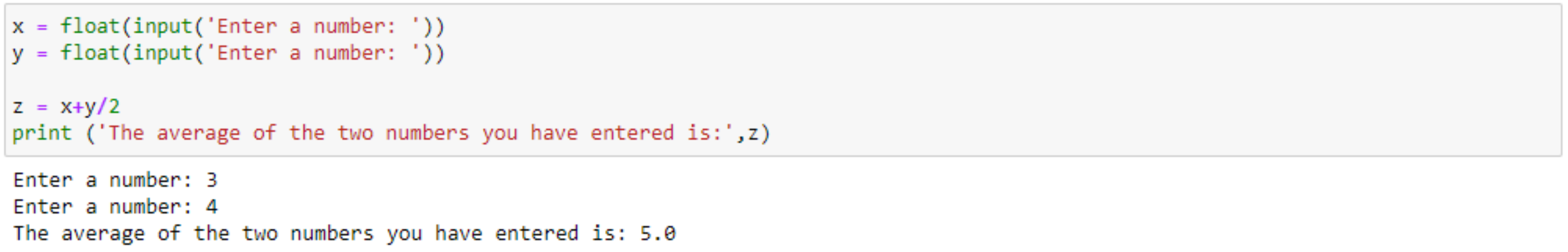
1) **Syntax errors** : Violation of grammar" rules. Easiest to fix. Python tells you at which line of your code the first error is detected.



```
print "hello world"
```

```
  File "<ipython-input-2-6d29d8fb337c>", line 1
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

2) **Logic errors** (also called semantic errors): logical errors cause the program to behave incorrectly. A program with logic errors can be run, but it does not operate as intended.

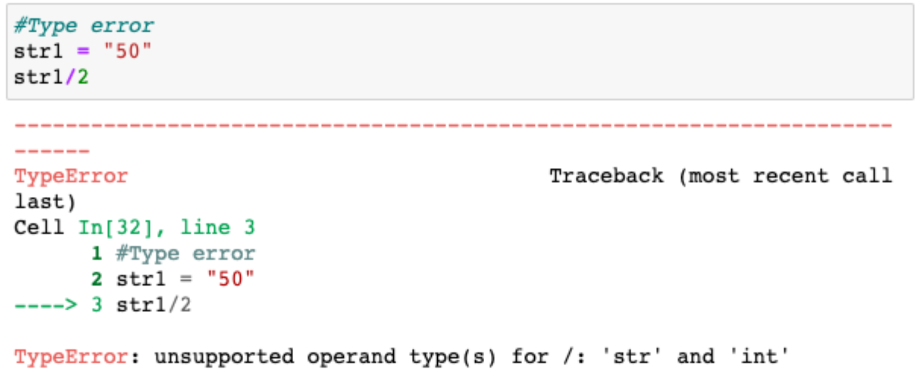


```
x = float(input('Enter a number: '))
y = float(input('Enter a number: '))

z = x+y/2
print ('The average of the two numbers you have entered is:',z)
```

```
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is: 5.0
```

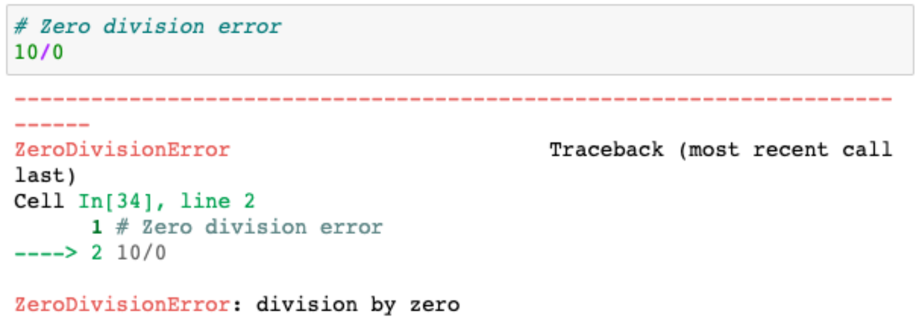
Built-in exceptions

- Python includes a collection of built-in exceptions that offer error messages and information, aiding in the process of debugging.



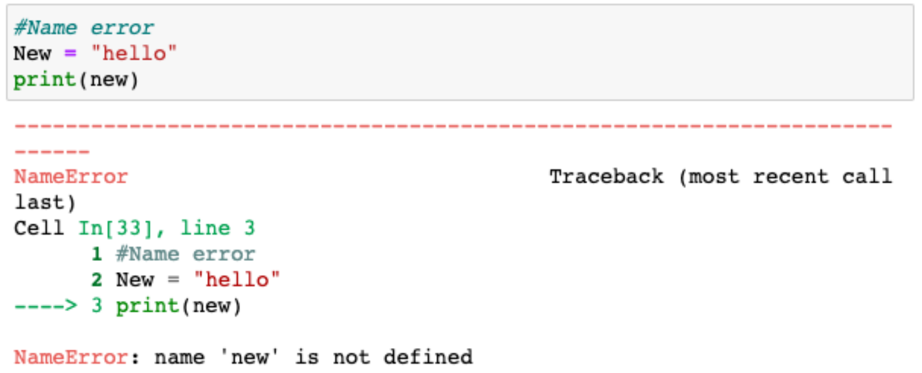
```
#Type error
str1 = "50"
str1/2
```

```
---------------------------------------------------------------------------
------
TypeError                                 Traceback (most recent call
last)
Cell In[32], line 3
      1 #Type error
      2 str1 = "50"
----> 3 str1/2

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```



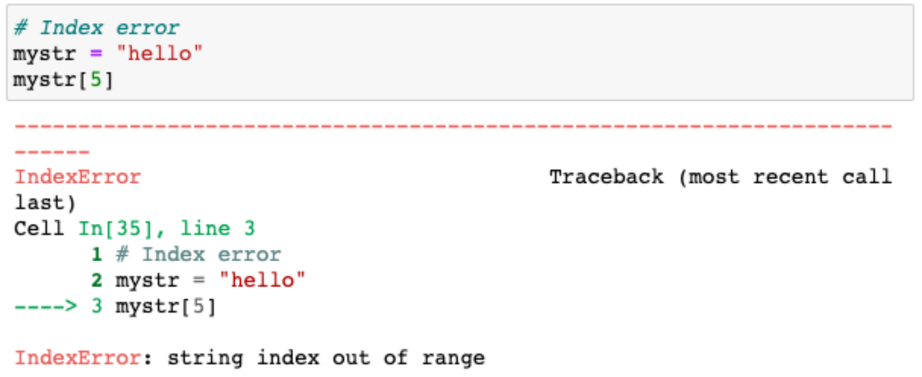
```
# Zero division error
10/0
```

```
---------------------------------------------------------------------------
------
ZeroDivisionError                         Traceback (most recent call
last)
Cell In[34], line 2
      1 # Zero division error
----> 2 10/0

ZeroDivisionError: division by zero
```



```
#Name error
New = "hello"
print(new)
```

```
---------------------------------------------------------------------------
------
NameError                                 Traceback (most recent call
last)
Cell In[33], line 3
      1 #Name error
      2 New = "hello"
----> 3 print(new)

NameError: name 'new' is not defined
```



```
# Index error
mystr = "hello"
mystr[5]
```

```
---------------------------------------------------------------------------
------
IndexError                                Traceback (most recent call
last)
Cell In[35], line 3
      1 # Index error
      2 mystr = "hello"
----> 3 mystr[5]

IndexError: string index out of range
```

See more types of errors https://docs.python.org/3/library/exceptions.html

Debugging

- What can you do about the errors? Debugging
  - Debugging is the process of finding and resolving bugs (problems that prevent correct operation) within computer programs.

- **Think**:
  - What kind of error is it: syntax error, logic error?
  - What information can you get from the error messages, or from the output of the program?
  - What kind of error could cause the problem you're seeing?
  - What did you change last, before the problem appeared?

- **Retreat**:
  - At some point, the best thing to do is back off, undoing recent changes, until you get back to a program that works and that you understand. Then you can start re-building.