KOÇ ÜNİVERSİTESİ

03 – Input, Processing, and Output

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

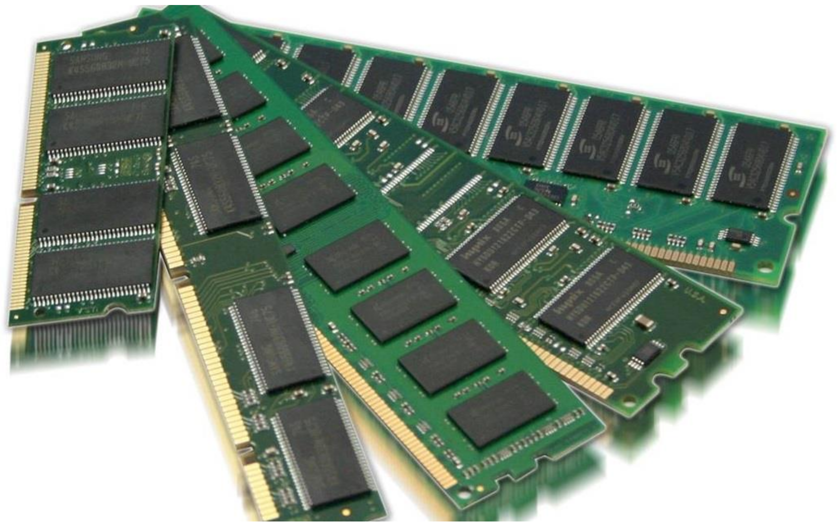
The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Variables

How do we store data?

- Data is stored in the main memory
- This is almost always the RAM



How do we store data in our program ?

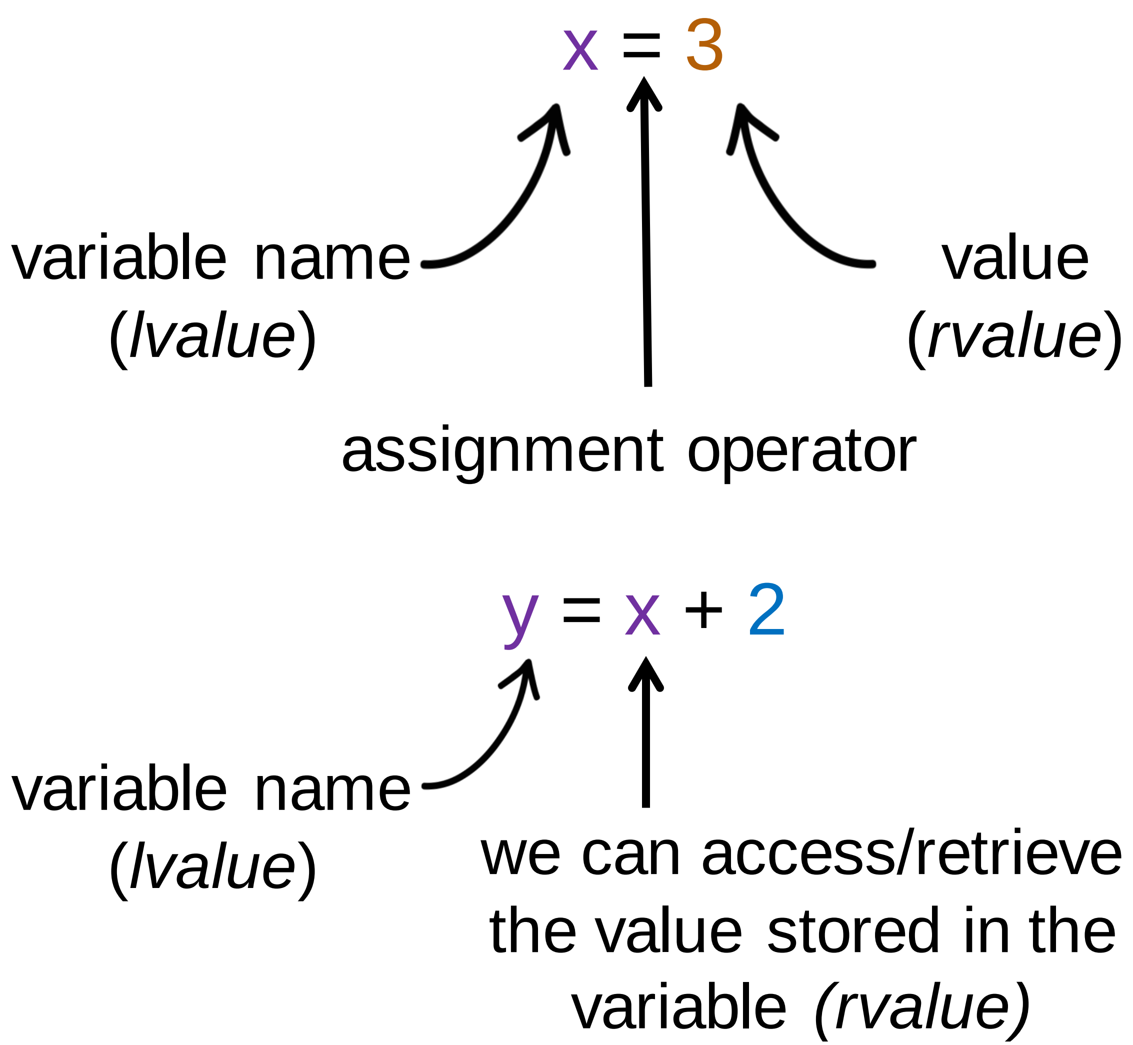
Variables !!!

Think of them as labels for data!

Definition

variable
a way for code to store information by associating a value with a name

An Example:



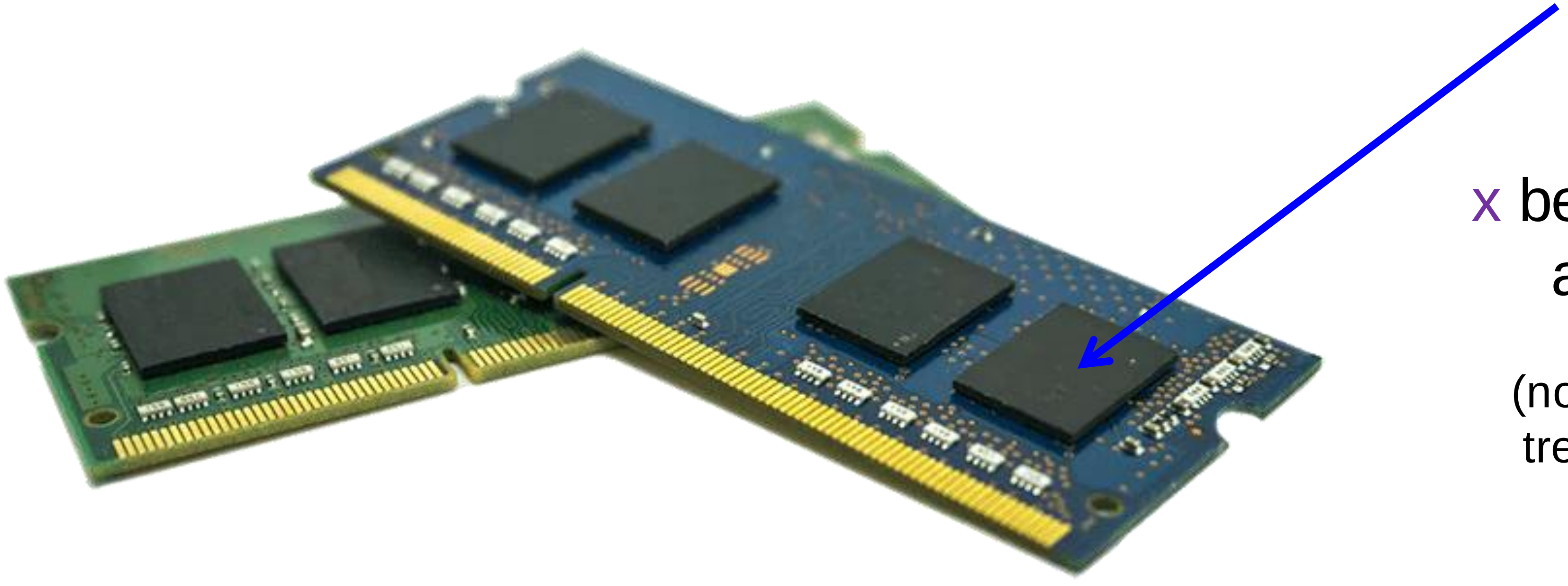
variable assignment
process of associating a name with a value (use the =); the variable is an ***lvalue***

variable retrieval
process of getting the value associated with a name; the variable is an ***rvalue***

Suitcase Analogy:

- When you store information in Python, it becomes ***a Python object***
  - Objects come in different sizes and types

x = 3

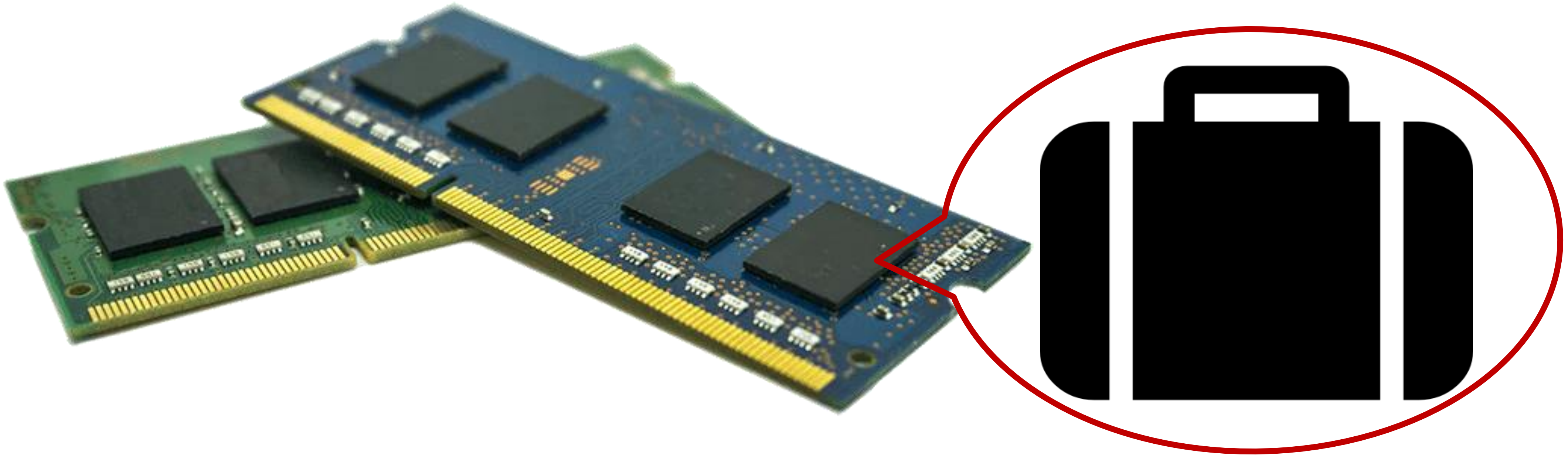


x becomes a Python object and is stored in RAM

(not all programming languages treat their variables as objects)

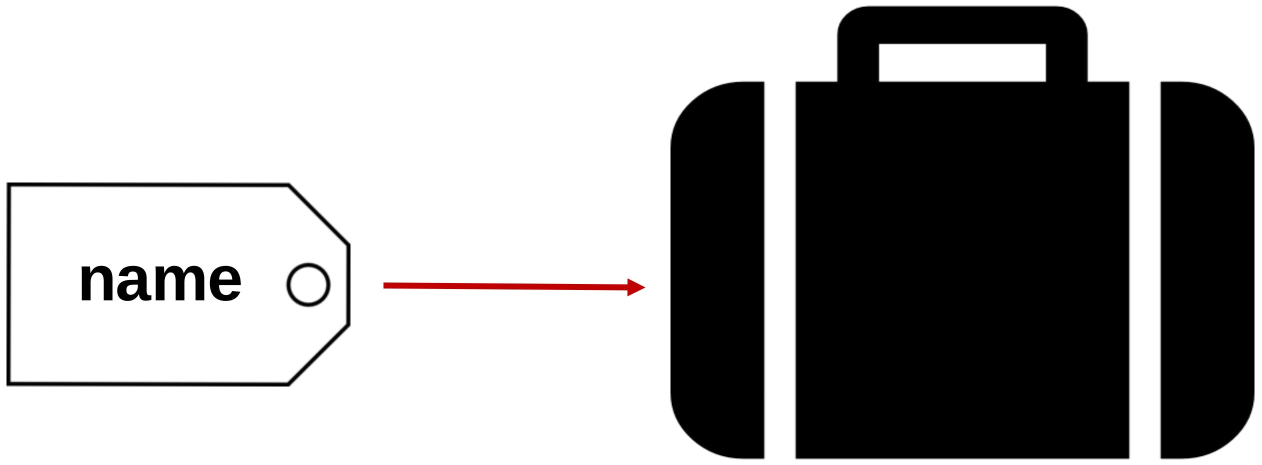
Suitcase Analogy:

- When you store information in Python, it becomes ***a Python object***
  - Objects come in different sizes and types
- Imagine a Python object as a suitcase stored in your computer’s memory, taking up different amounts of RAM depending on what you’re storing



Suitcase Analogy:

- When you store information in Python, it becomes ***a Python object***
  - Objects come in different sizes and types (more on these later)
- Imagine a Python object as a suitcase stored in your computer's memory, taking up different amounts of RAM depending on what you're storing
- Continuing with the analogy, a **variable** is a tag for your suitcase so that you can refer to it with a **name**!



Variables in Python

- A **variable** is a name that represents a value stored in the computer memory
  - Used to access and manipulate data stored in memory
  - A variable ***references*** the value it represents
  - In other words, in Python, each variable **stores a memory address** (the memory location in which the data is stored)

- An **assignment statement** is used to create a variable and make it ***reference* the data**
  - General format is `variable = expression`
  - `Expression` could include another variable (*rvalue*) or a literal constant or any function of these.
  - Equal sign (=) is the assignment operator

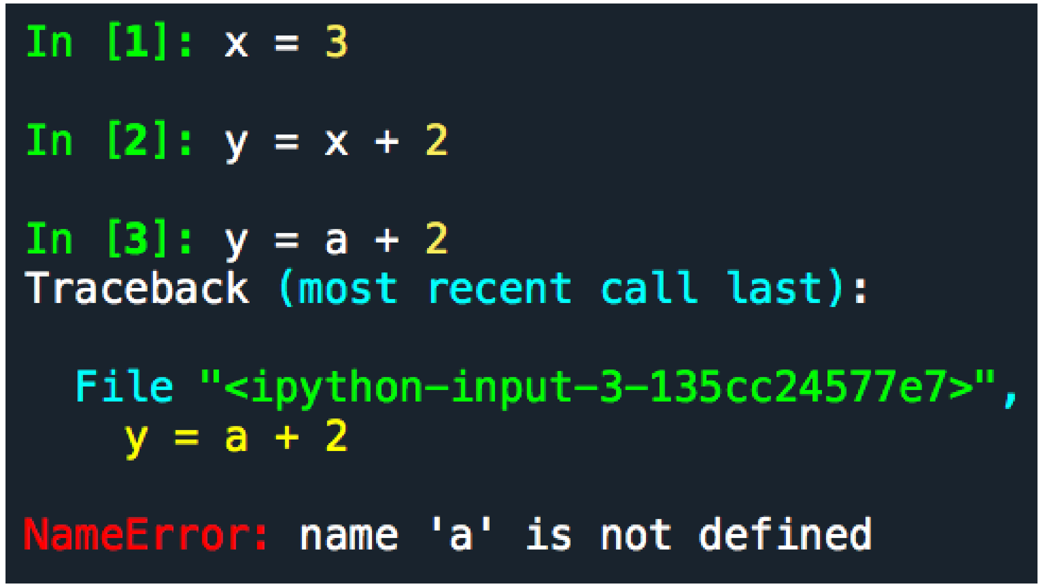
```
x = 3
y = (x + 4) * 5
x = y / x + 1
```

Variables in Python

- When assigning, a variable **receiving** the value must be on the **left** side (i.e., the variable should be an *lvalue)*

```
x = 3
```

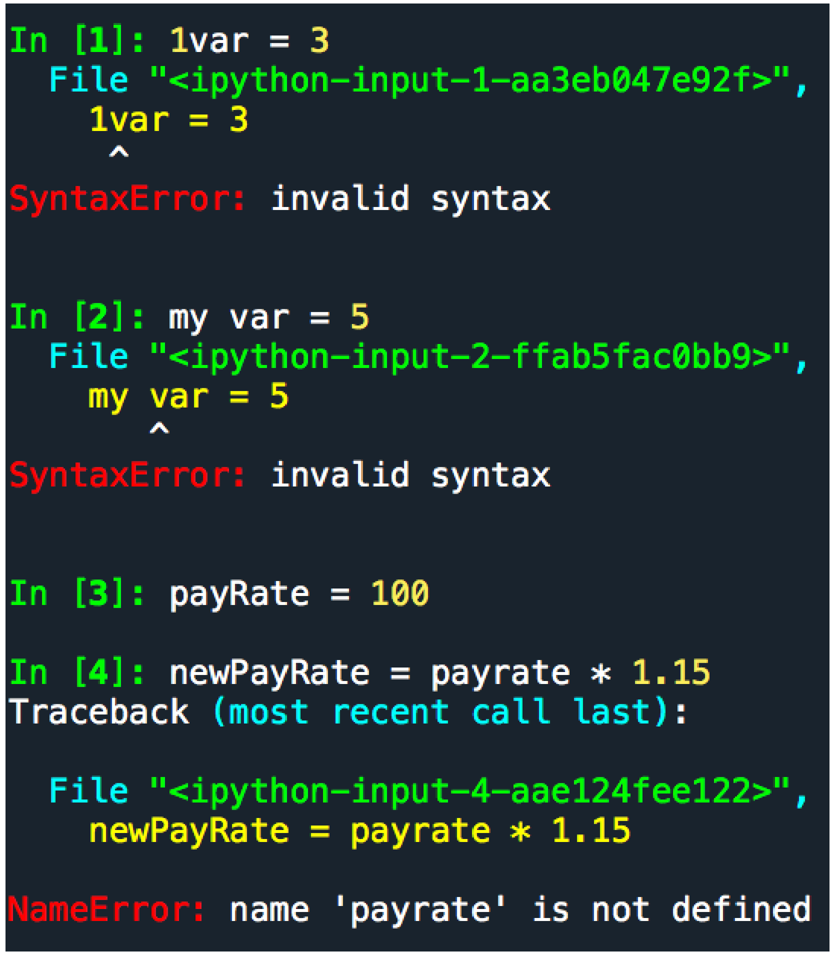
- You can only use a variable as an *rvalue* if a value has already been **assigned** to it
  - Otherwise, you get an error!



```
In [1]: x = 3

In [2]: y = x + 2

In [3]: y = a + 2
Traceback (most recent call last):

  File "<ipython-input-3-135cc24577e7>",
    y = a + 2

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

Rules for Variable Naming

- You **can only** use digits [0-9], letters [*a-z]* or [*A-Z*] or an underscore character '_'
  - *Legal:* **Variable_1, FirstName**
  - *Illegal:* **!variable, variable#2, first-name**
- You **cannot** start with a number
  - *Illegal:* ***1variable = 3***
- You **cannot** leave spaces
  - *Illegal:* **my variable = 3**
- You **cannot** use Python's keywords
  - *Illegal:* **import, break, return, for, break, *etc.***
- Python is ***case-sensitive***
  - Lowercase and uppercase letters are distinct
  - e.g., payRate is not the same as payrate or Payrate

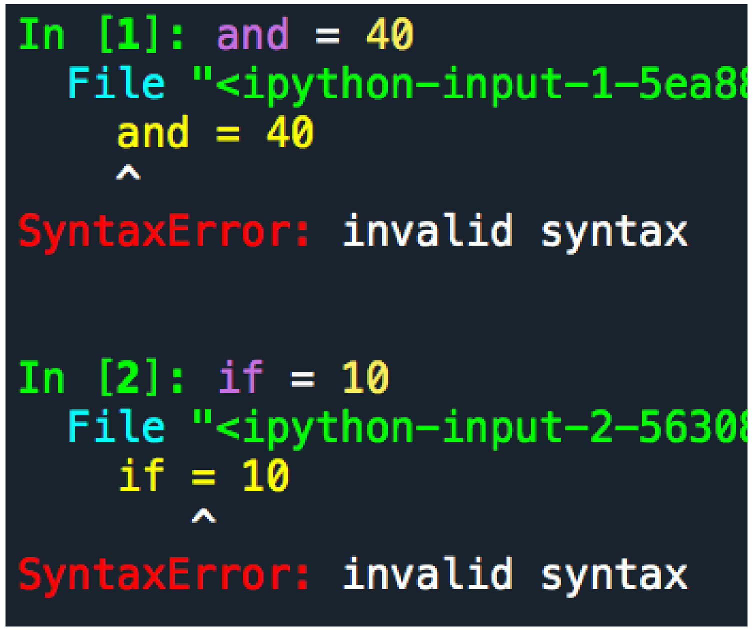


```
In [1]: 1var = 3
  File "<ipython-input-1-aa3eb047e92f>",
    1var = 3
     ^
SyntaxError: invalid syntax


In [2]: my var = 5
  File "<ipython-input-2-ffab5fac0bb9>",
    my var = 5
       ^
SyntaxError: invalid syntax


In [3]: payRate = 100

In [4]: newPayRate = payrate * 1.15
Traceback (most recent call last):

  File "<ipython-input-4-aae124fee122>",
    newPayRate = payrate * 1.15

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

Python Keywords

- Each keyword has a predefined functionality (*we will discuss them later*)
- Thus, you cannot use them as variable names
- No need to memorize!

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



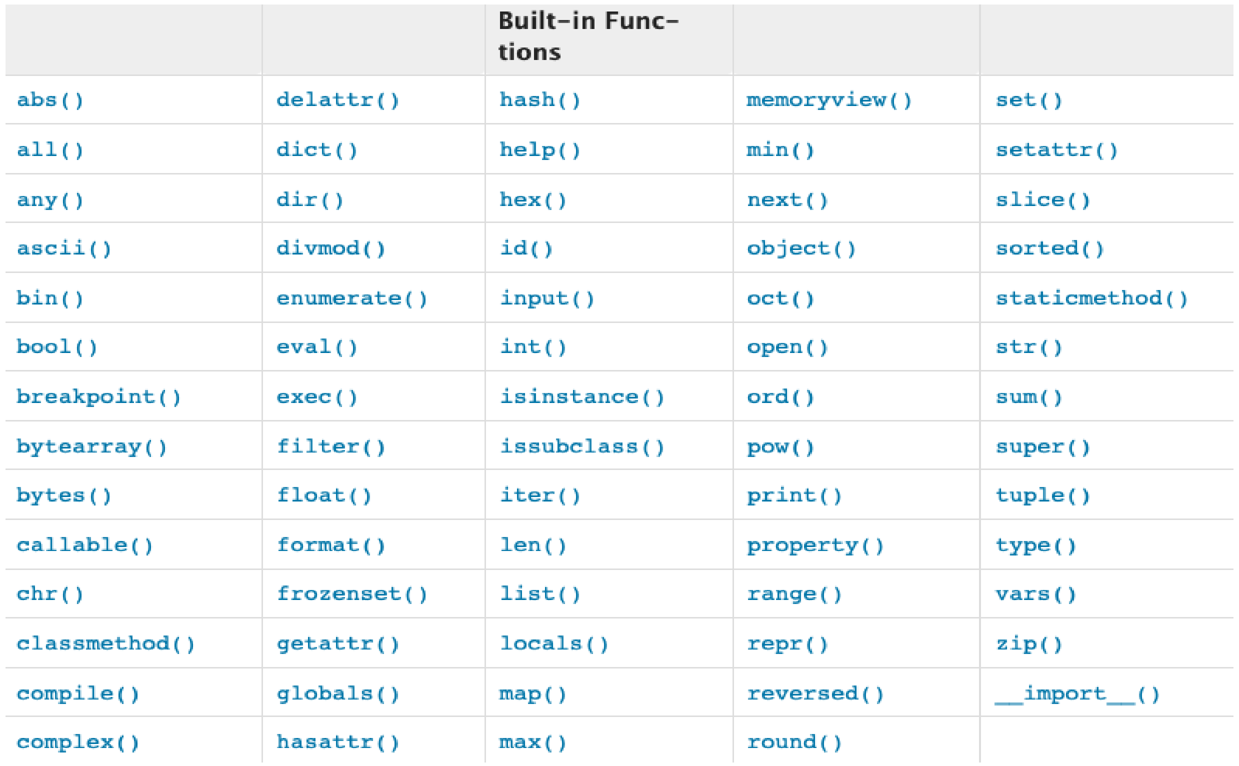
```
In [1]: and = 40
  File "<ipython-input-1-5ea8
    and = 40
    ^
SyntaxError: invalid syntax


In [2]: if = 10
  File "<ipython-input-2-5630
    if = 10
       ^
SyntaxError: invalid syntax
```

Pop Quiz: Legal or Illegal?

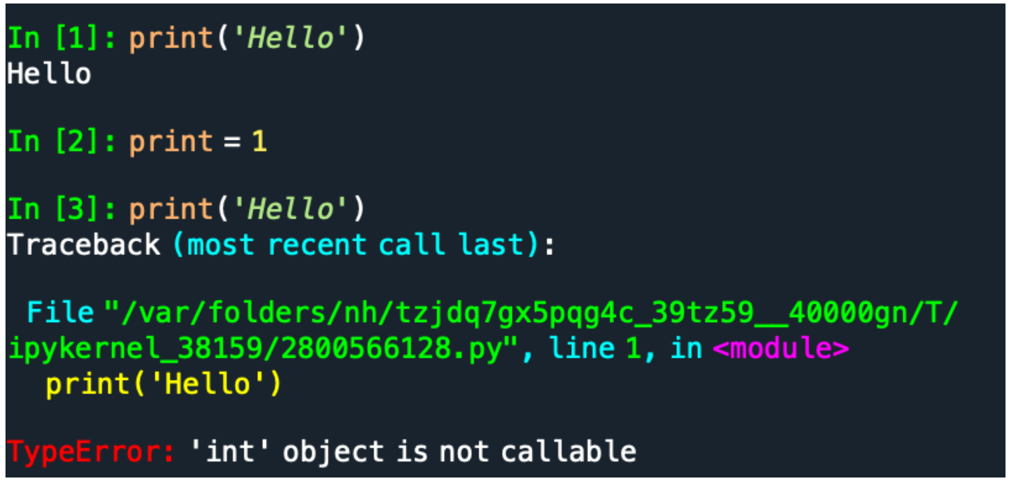
Variable Name	
hours_per_day	**legal**
January2021	**legal**
5thOfDecember	**illegal**
Variable#3	**illegal**
myNumber	**legal**
_myNumber	**legal**
-myNumber	**illegal**
False	**illegal**
while	**illegal**

Python Built-in Function Names

- Python allows the built-in function names to be used as variable names.
  But in that case the respective function can no longer be called
- Avoid using built-in function names as variable names
- No need to memorize!



		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	



```
In [1]: print('Hello')
Hello

In [2]: print = 1

In [3]: print('Hello')
Traceback (most recent call last):

 File "/var/folders/nh/tzjdq7gx5pqg4c_39tz59__40000gn/T/
ipykernel_38159/2800566128.py", line 1, in <module>
  print('Hello')

TypeError: 'int' object is not callable
```

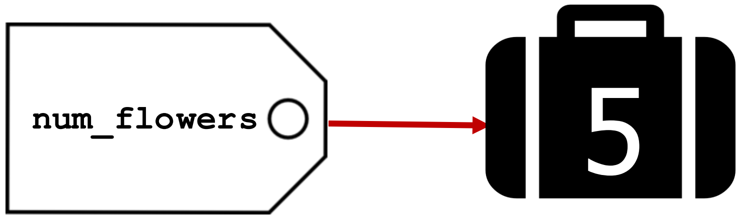
Another Example

- Suppose you have 5 flowers in your garden. You pick 2 flowers to give to your friend.
- Write a code fragment that initializes the number of flowers in your garden with 5, and decrements its value by the number of picked flowers

```
num_flowers = 5
```

variable name

value (literal constant or simply literal)



Variable assignment
(attaching the tag)

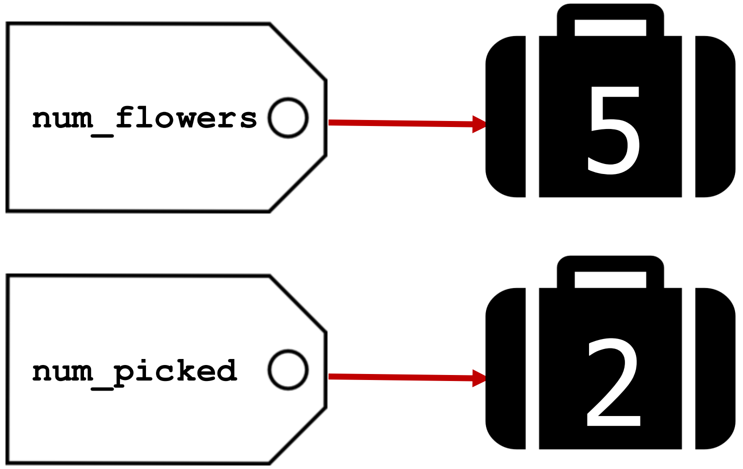
Another Example

- Write a code fragment that initializes the number of flowers in your garden with 5, and decrements its value by the number of picked flowers

```
num_flowers = 5

num_picked = 2

num_flowers = num_flowers - num_picked
```

The right side of the assignment **always** gets evaluated first

1. Retrieve values of the variables (5 and 2 respectively)
2. Evaluate the *expression* (the result is 3)
3. **THEN WHAT?**



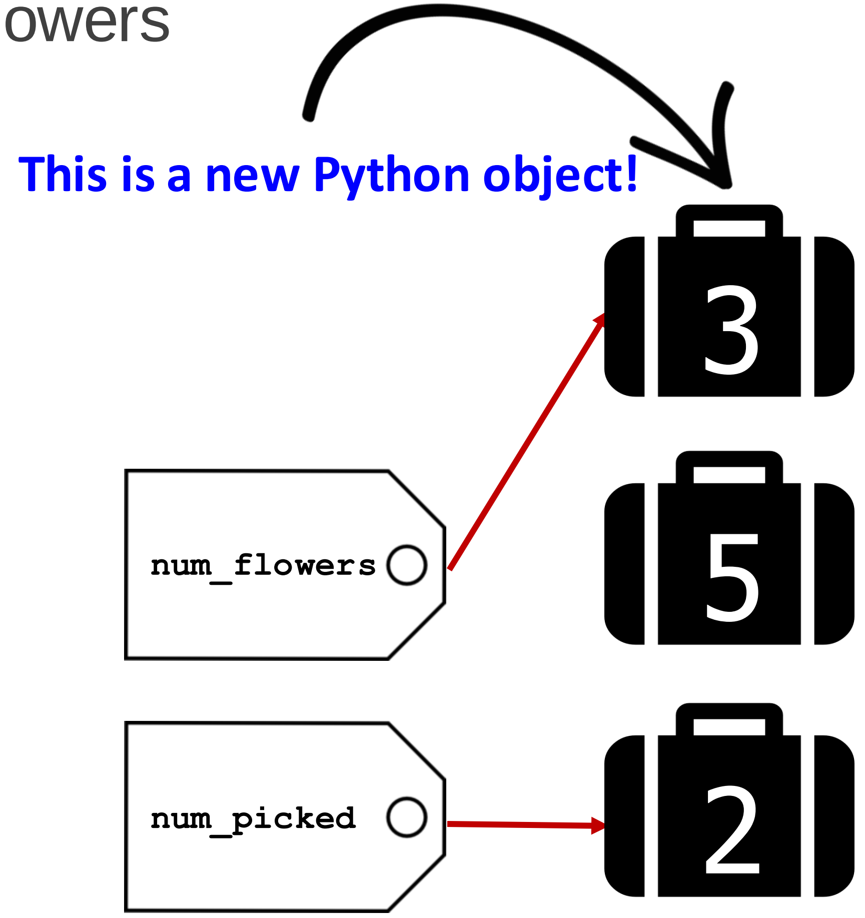
Another Example

- Write a code fragment that initializes the number of flowers in your garden with 5, and decrements its value by the number of picked flowers

```
num_flowers = 5

num_picked = 2

num_flowers = num_flowers - num_picked
```

The right side of the assignment **always** gets evaluated first

1. Retrieve values of the variables (5 and 2 respectively)
2. Evaluate the *expression* (the result is 3)
3. **The expression result is assigned to the LHS variable**

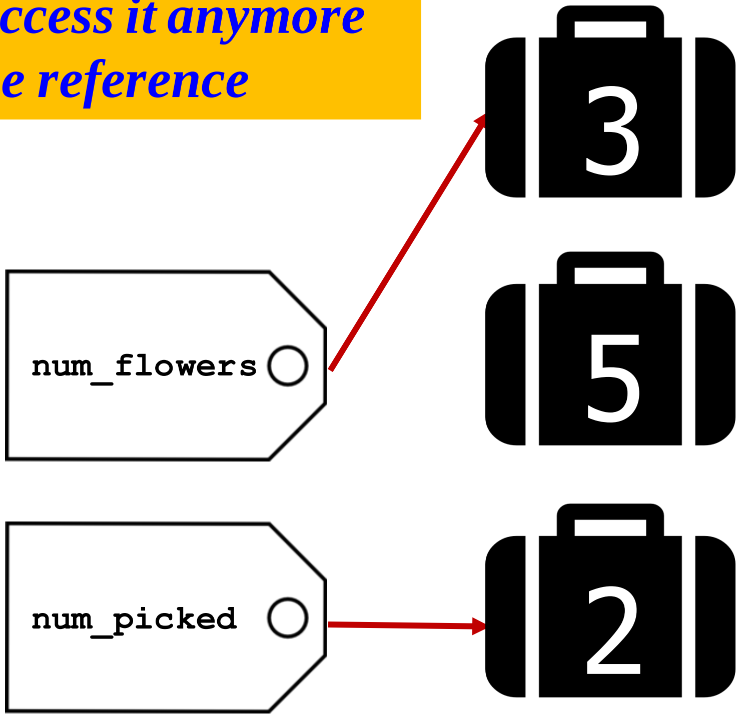


Another Example

- Write a code fragment that initializes the number of flowers in your garden with 5, and decrements its value by the number of picked flowers

```
num_flowers = 5

num_picked = 2

num_flowers = num_flowers - num_picked
```

What happened to 5?
You cannot access it anymore as you lost the reference



The right side of the assignment **always** gets evaluated first

1. Retrieve values of the variables (5 and 2 respectively)
2. Evaluate the *expression* (the result is 3)
3. **The expression result is assigned to the LHS variable**

Summary

- Variables have a **name** and are associated with a **value**
  - Recommended to use descriptive variable names
  - All names should have a valid syntax
- Variable **assignment** is the process of associating a value with the name (use the equals sign =)
  - This is how we **create** and **update** variables!
- Retrieval is the process of getting the value associated with the name (use the variable's name)
  - This is how we **access** the value of the variables!

```
x = 3
y = (x + 4) * 5
x = y / x + 1
```

lvalue *rvalue*

Exercise

A(n) ____________ makes a variable reference a value in the computer's memory.

a. variable declaration

$\boxed{\text{b. assignment statement}}$

c. math expression

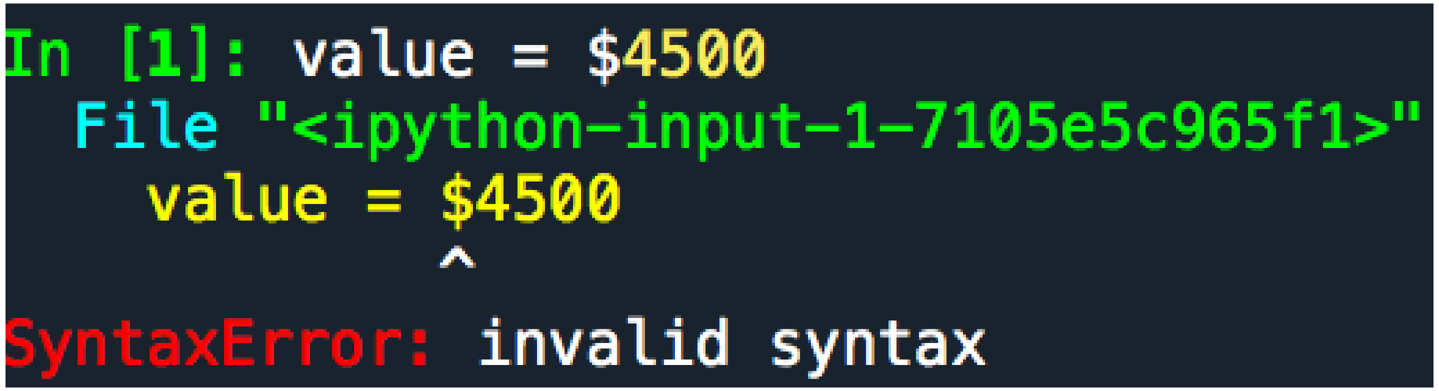
d. string literal

```
variable = expression

x = 3 (literal)
a = x (another variable)
a = x * 5 + 3 (mathematical expression)
y = a + 30 (mathematical expression)
```

Exercise

What will happen if you try to perform the following?

value = $4500



```
In [1]: value = $4500
  File "<ipython-input-1-7105e5c965f1>"
    value = $4500
            ^
SyntaxError: invalid syntax
```

Exercise: Fibonacci Sequence

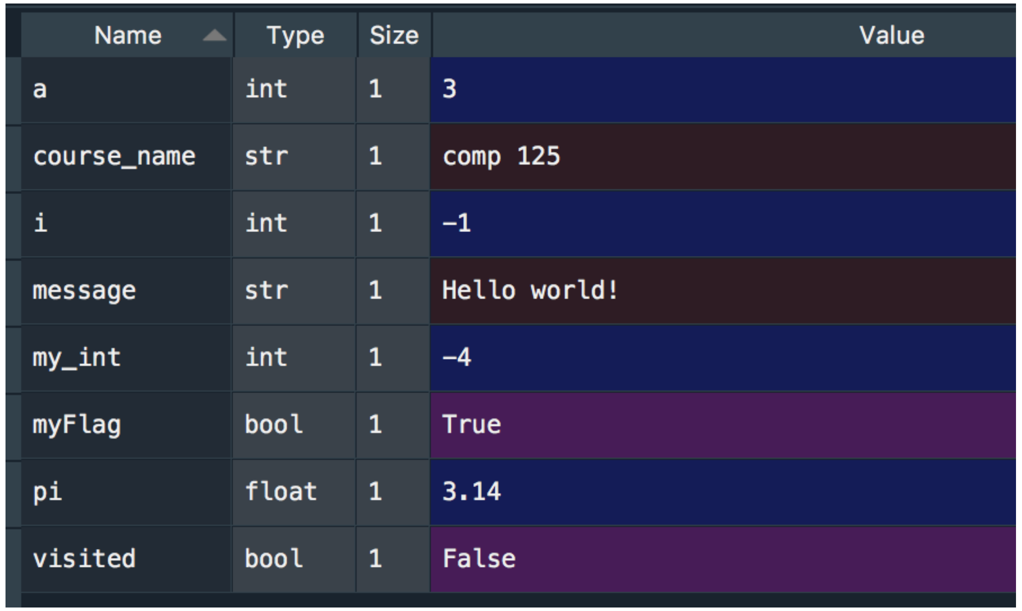
- Let's calculate the Fibbonacci sequence.
- Starts with two ones:
  - 1 1
- Next number is obtained by summing up the last two items in the sequence
  - 1 1 2
  - 1 1 2 3
  - 1 1 2 3 5
  - 1 1 2 3 5 8
  - 1 1 2 3 5 8 13
- Let's create two variables for the last (**first_item**) and 2nd from the last (**second_item**) items in the sequence
- Calculate the new item and assign it to the variable **new_item**.
- Write a code to calculate the Fibonacci sequence.

Data Types

Data Types

- strings: `str`
  - `message = 'Hello world!'`
  - `course_name = 'comp 125'`
- integers: `int`
  - `a = 3`
  - `my_int = -4`
- real numbers (floating point numbers): `float`
  - `pi = 3.14`
  - `i = -1.0`
- Boolean with two possible special values: `bool`
  - `my_flag = True`
  - `visited = False`

Numeric Data Types



Name	Type	Size	Value
a	int	1	3
course_name	str	1	comp 125
i	int	1	-1
message	str	1	Hello world!
my_int	int	1	-4
myFlag	bool	1	True
pi	float	1	3.14
visited	bool	1	False

Exercise

You were asked to write a program for a doctor's office.

What **type** would you use to store the following?

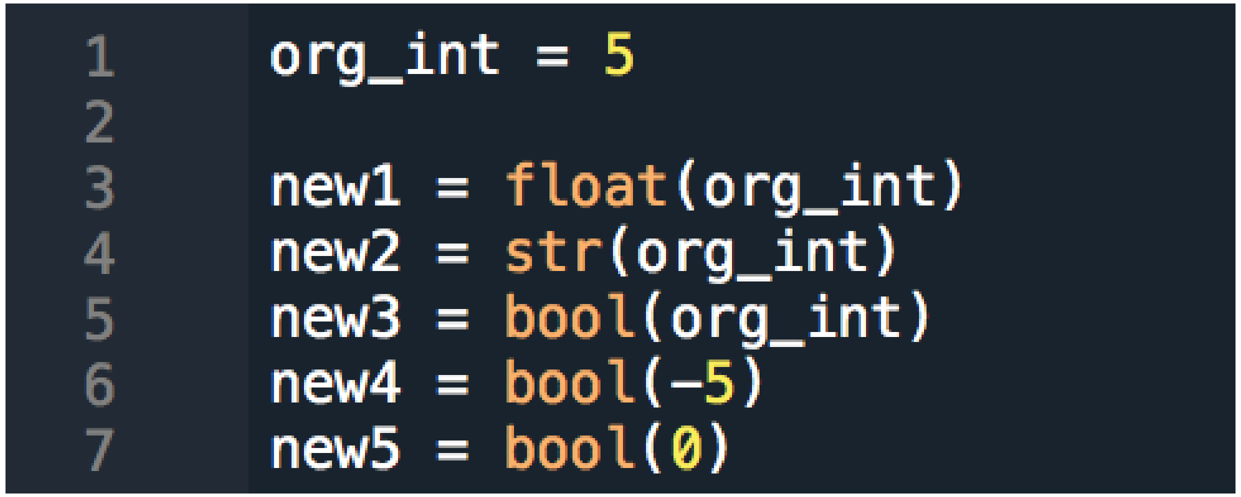
o The patient's weight	→	**float**
o The number of days since the patient's last visit	→	**int**
o The patient's temperature	→	**float**
o If the patient has had their flu shot	→	**bool**
o The patient's number of children	→	**int**
o The name of the patient	→	**str**

Exercise

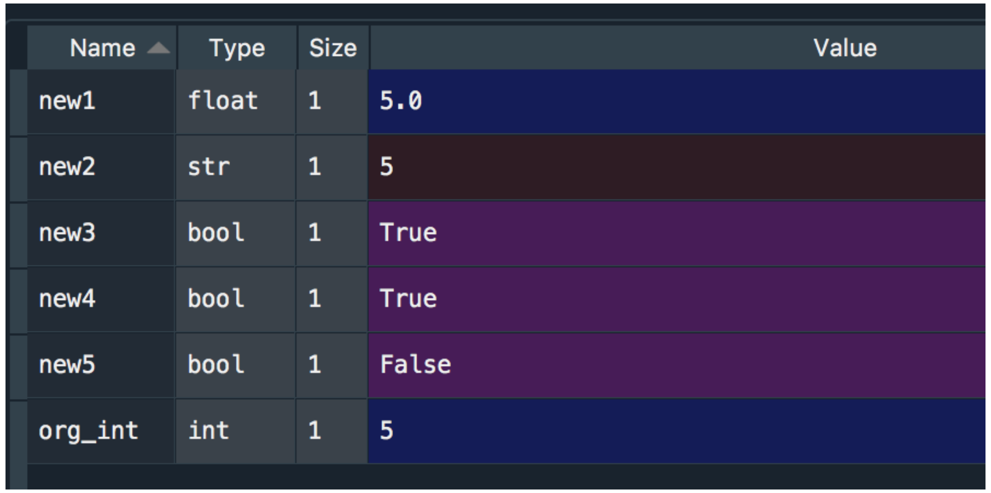
After these statements execute, what is the Python data type of the values referenced by each variable?

value1 = 26	→	int
value2 = 43.4	→	float
value3 = 7.0	→	float
value4 = 7	→	int
value5 = 'abc'	→	str
value6 = True	→	bool
value7 = 'True'	→	str

Data Type Conversion - (from int)



```
org_int = 5

new1 = float(org_int)
new2 = str(org_int)
new3 = bool(org_int)
new4 = bool(-5)
new5 = bool(0)
```



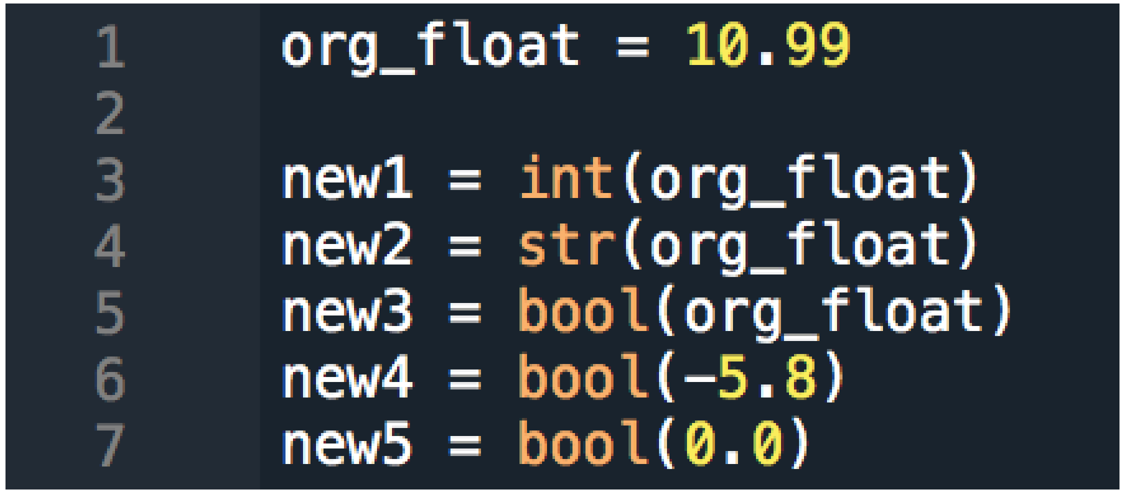
Name	Type	Size	Value
new1	float	1	5.0
new2	str	1	5
new3	bool	1	True
new4	bool	1	True
new5	bool	1	False
org_int	int	1	5

When converting from an integer to a Boolean

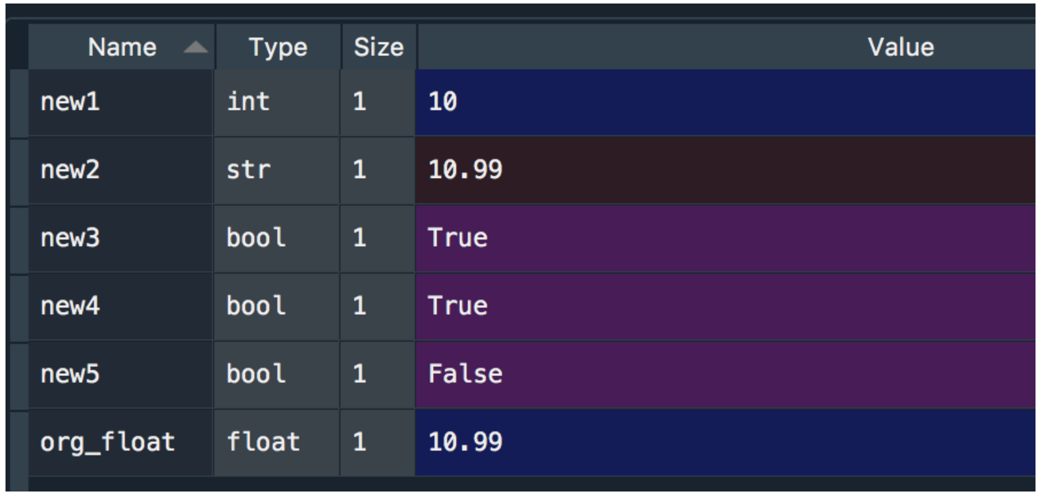
- Only zero is considered as False
- All non-zero values (both positive and negative ones) are considered as True

The original value and type of the variable (in this case, those of the variable org_int) remain unchanged

Data Type Conversion - (from float)



```
org_float = 10.99

new1 = int(org_float)
new2 = str(org_float)
new3 = bool(org_float)
new4 = bool(-5.8)
new5 = bool(0.0)
```



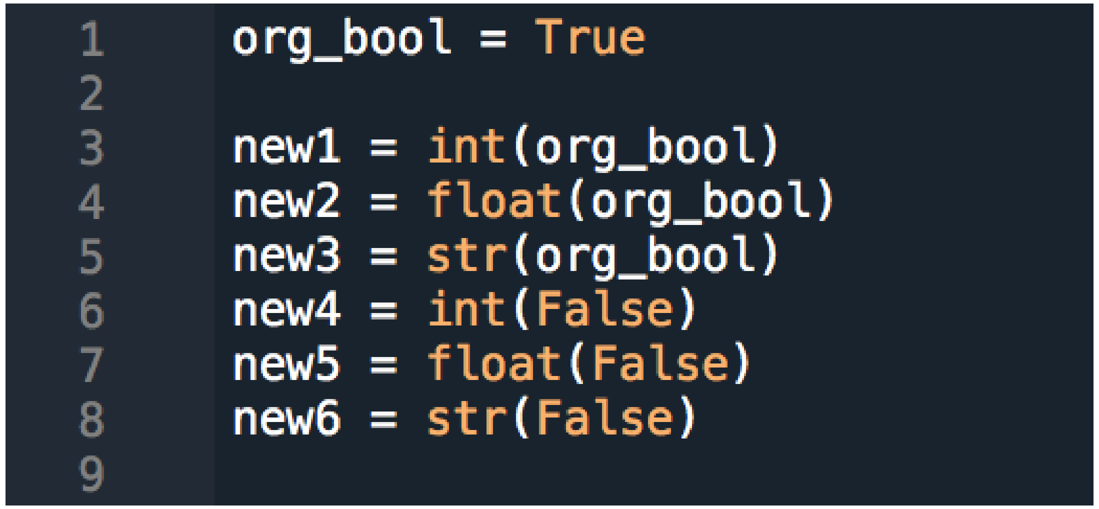
Name	Type	Size	Value
new1	int	1	10
new2	str	1	10.99
new3	bool	1	True
new4	bool	1	True
new5	bool	1	False
org_float	float	1	10.99

When converting from a float to an integer

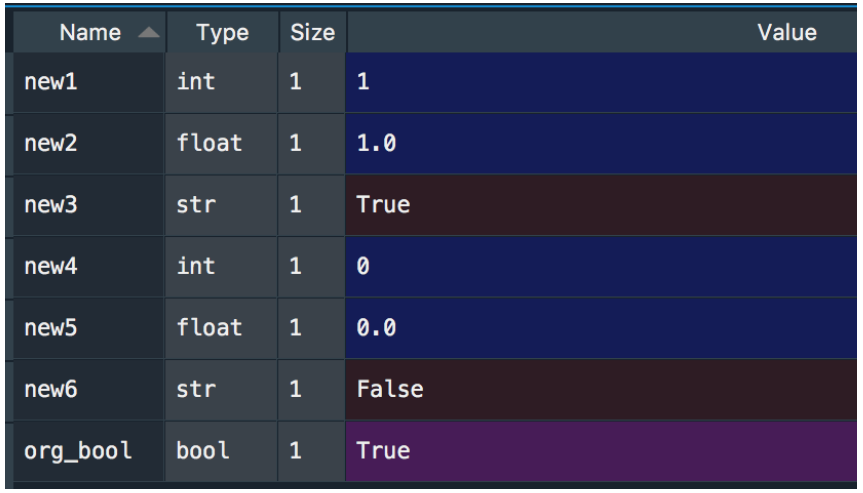
- The value is truncated (no rounding operation)

Likewise, the original value and type of the variable (org_float) remain unchanged

Data Type Conversion - (from bool)



```
org_bool = True

new1 = int(org_bool)
new2 = float(org_bool)
new3 = str(org_bool)
new4 = int(False)
new5 = float(False)
new6 = str(False)
```



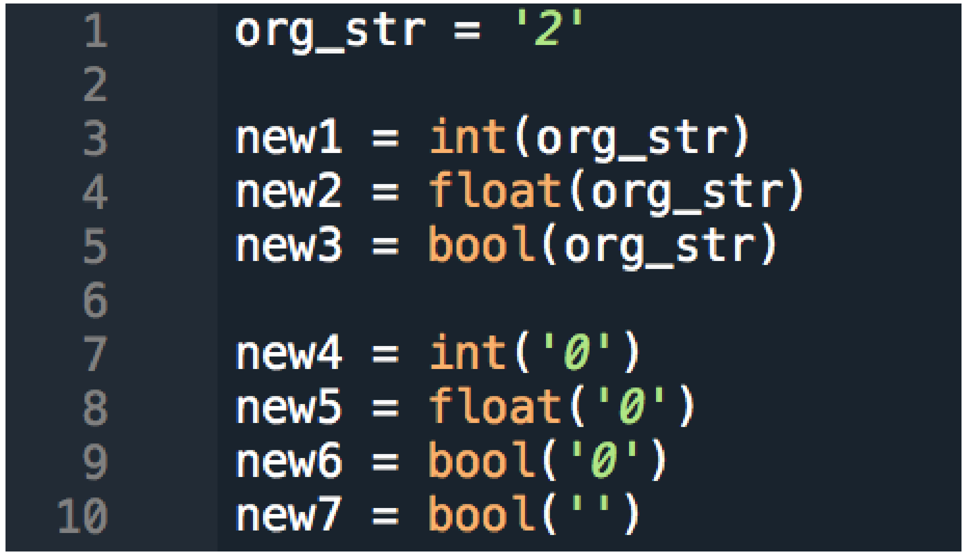
Name	Type	Size	Value
new1	int	1	1
new2	float	1	1.0
new3	str	1	True
new4	int	1	0
new5	float	1	0.0
new6	str	1	False
org_bool	bool	1	True

When converting from a Boolean to an integer or a float

- True is considered as 1 or 1.0 (no other values are possible)
- False is considered as 0 or 0.0

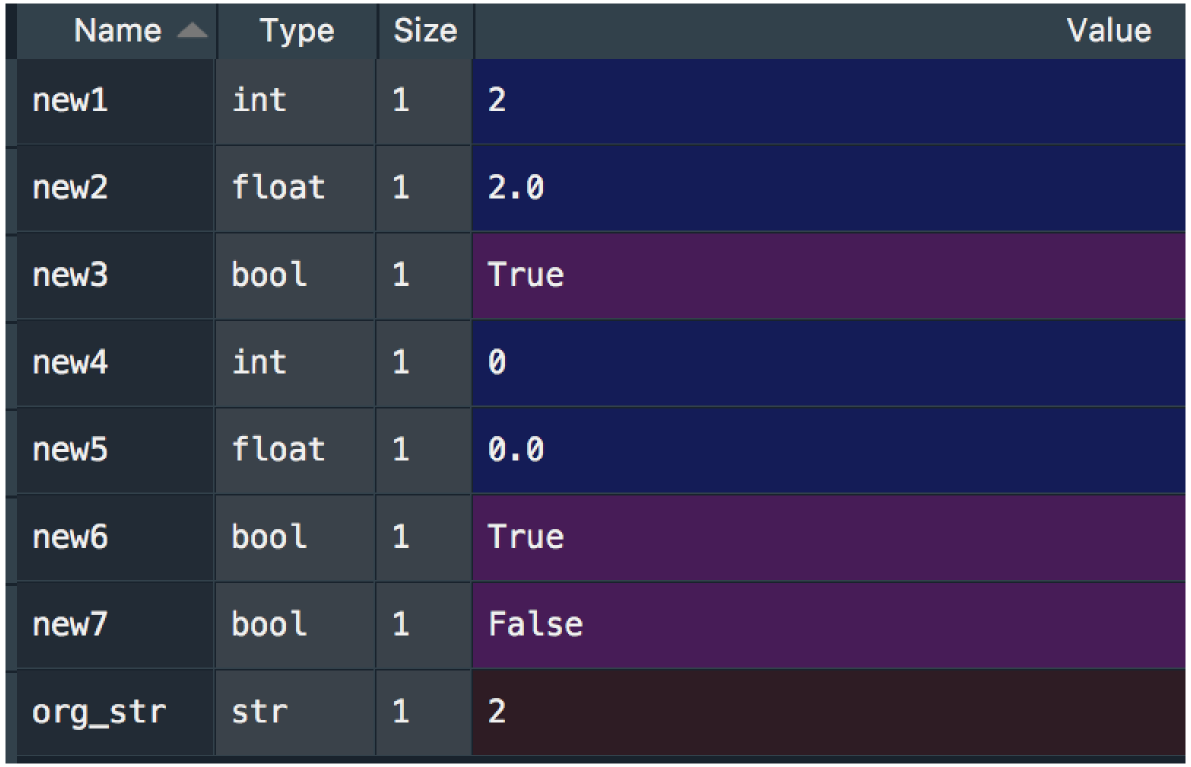
Likewise, the original value and type of the variable (org_bool) remain unchanged

Data Type Conversion - (from str)



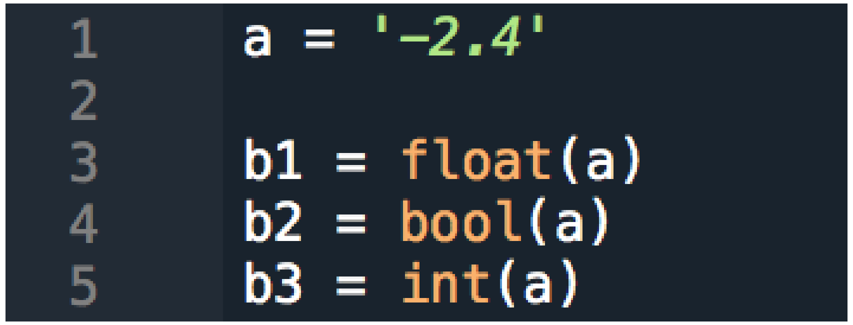
```
org_str = '2'

new1 = int(org_str)
new2 = float(org_str)
new3 = bool(org_str)

new4 = int('0')
new5 = float('0')
new6 = bool('0')
new7 = bool('')
```

Check the values of the new6 and new7 variables after the conversion

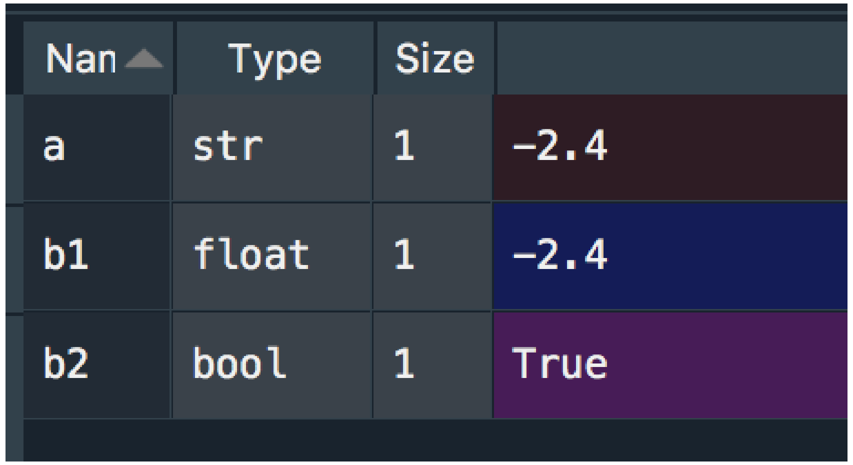


Name	Type	Size	Value
new1	int	1	2
new2	float	1	2.0
new3	bool	1	True
new4	int	1	0
new5	float	1	0.0
new6	bool	1	True
new7	bool	1	False
org_str	str	1	2

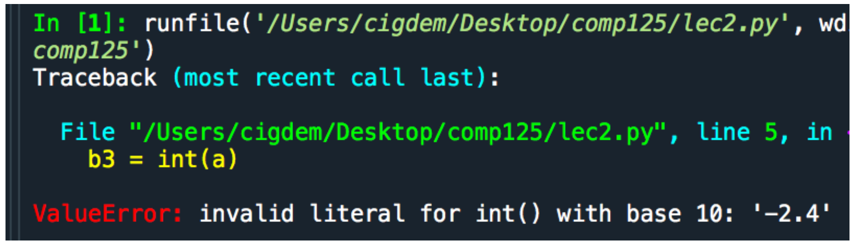
Data Type Conversion - (from str)



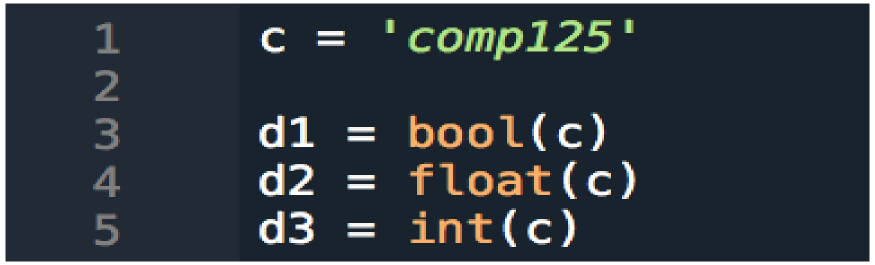
```python
a = '-2.4'

b1 = float(a)
b2 = bool(a)
b3 = int(a)
```



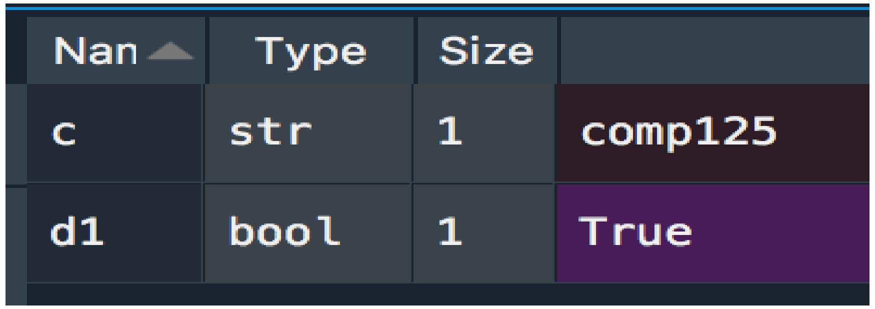
Nan	Type	Size	
a	str	1	-2.4
b1	float	1	-2.4
b2	bool	1	True



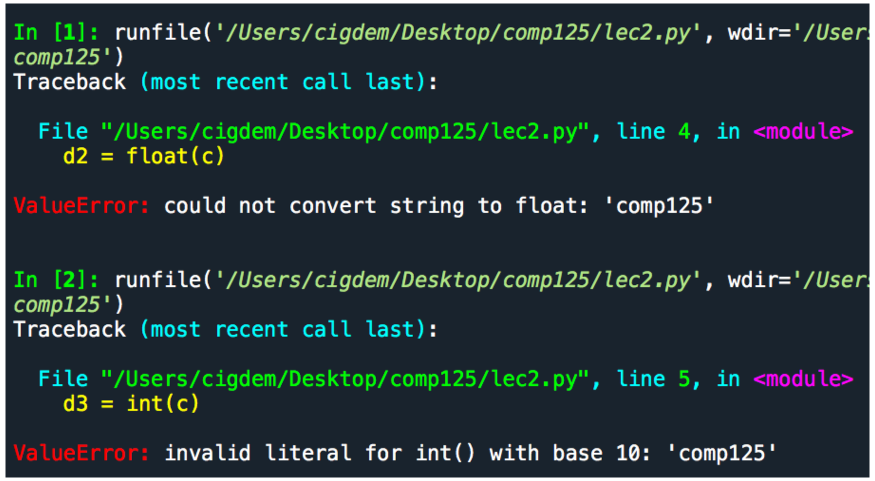
```
In [1]: runfile('/Users/cigdem/Desktop/comp125/lec2.py', wd
comp125')
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/comp125/lec2.py", line 5, in
    b3 = int(a)

ValueError: invalid literal for int() with base 10: '-2.4'
```



```python
c = 'comp125'

d1 = bool(c)
d2 = float(c)
d3 = int(c)
```



Nan	Type	Size	
c	str	1	comp125
d1	bool	1	True

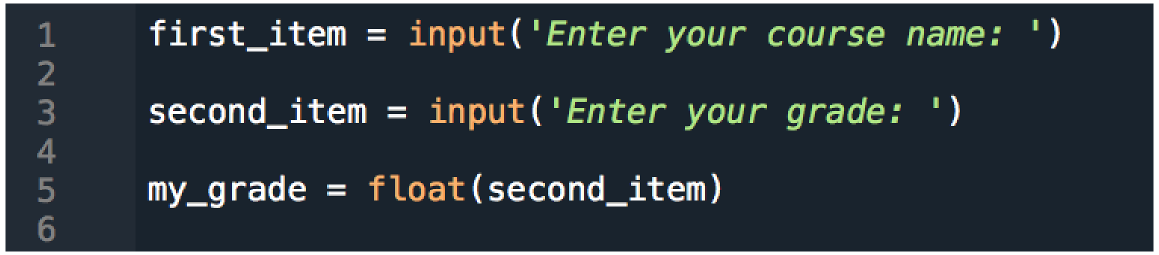


```
In [1]: runfile('/Users/cigdem/Desktop/comp125/lec2.py', wdir='/User
comp125')
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/comp125/lec2.py", line 4, in <module>
    d2 = float(c)

ValueError: could not convert string to float: 'comp125'


In [2]: runfile('/Users/cigdem/Desktop/comp125/lec2.py', wdir='/User
comp125')
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/comp125/lec2.py", line 5, in <module>
    d3 = int(c)

ValueError: invalid literal for int() with base 10: 'comp125'
```

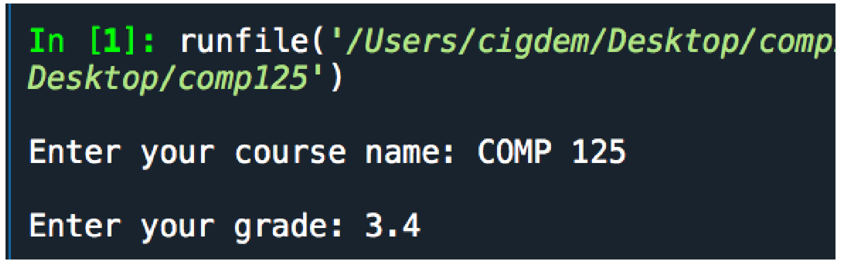
Input and Output

Reading Input from Keyboard

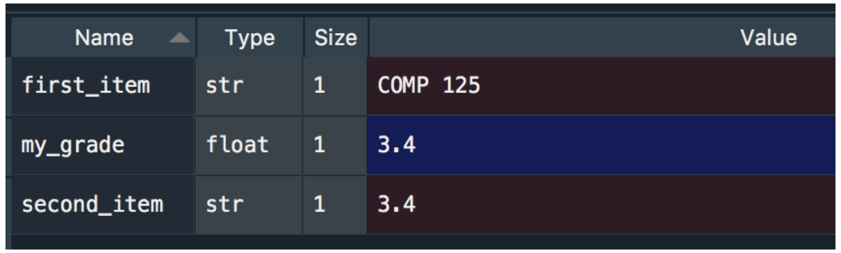
```
item = input(prompt_message)
```



```
first_item = input('Enter your course name: ')

second_item = input('Enter your grade: ')

my_grade = float(second_item)
```



```
In [1]: runfile('/Users/cigdem/Desktop/comp
Desktop/comp125')

Enter your course name: COMP 125

Enter your grade: 3.4
```



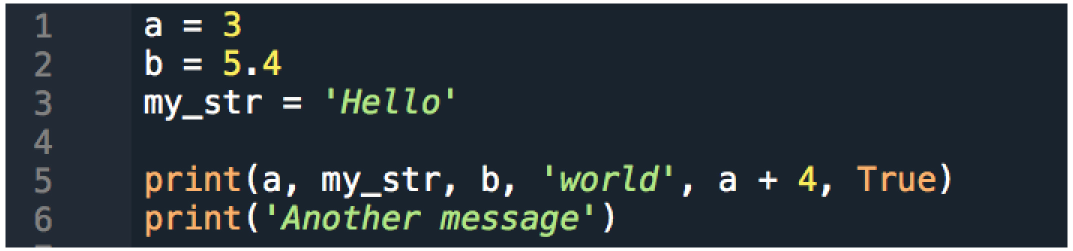
Name	Type	Size	Value
first_item	str	1	COMP 125
my_grade	float	1	3.4
second_item	str	1	3.4

- `input` function prompts a message on the screen and tells the program to stop until the user keys in the data
- This function always returns the user's input as a string, even if the user enters numeric data (`item` will have a data type of `str`)
- Thus, if you need numeric data, you have use data type of conversion, using `int(item)` or `float(item)`

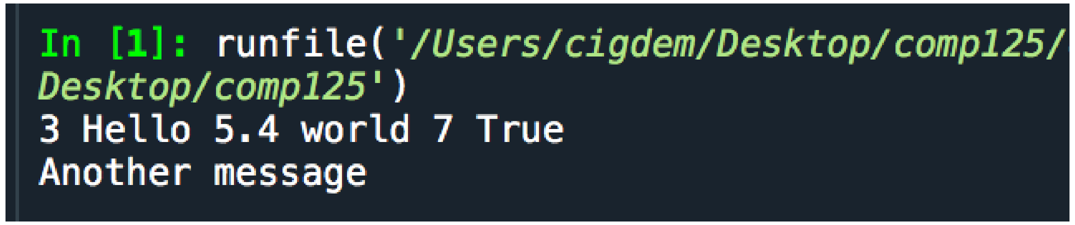
Displaying Output to Screen

`print(message)`

- **`print`** function displays the specified message on the screen (or other standard output devices)
- **`message`** is a comma separated list that contains one or more literals, variables, and expressions
- Otherwise specified, each **`print`** function ends with a new line



```python
a = 3
b = 5.4
my_str = 'Hello'

print(a, my_str, b, 'world', a + 4, True)
print('Another message')
```



```
In [1]: runfile('/Users/cigdem/Desktop/comp125/
Desktop/comp125')
3 Hello 5.4 world 7 True
Another message
```

Escape Characters

- Appear inside a string literal and are preceded with a backslash (\).
- The most important one we will use is the **newline escape character** (\n).

```
print('One\nTwo\nThree')
```

```
One
Two
Three
```

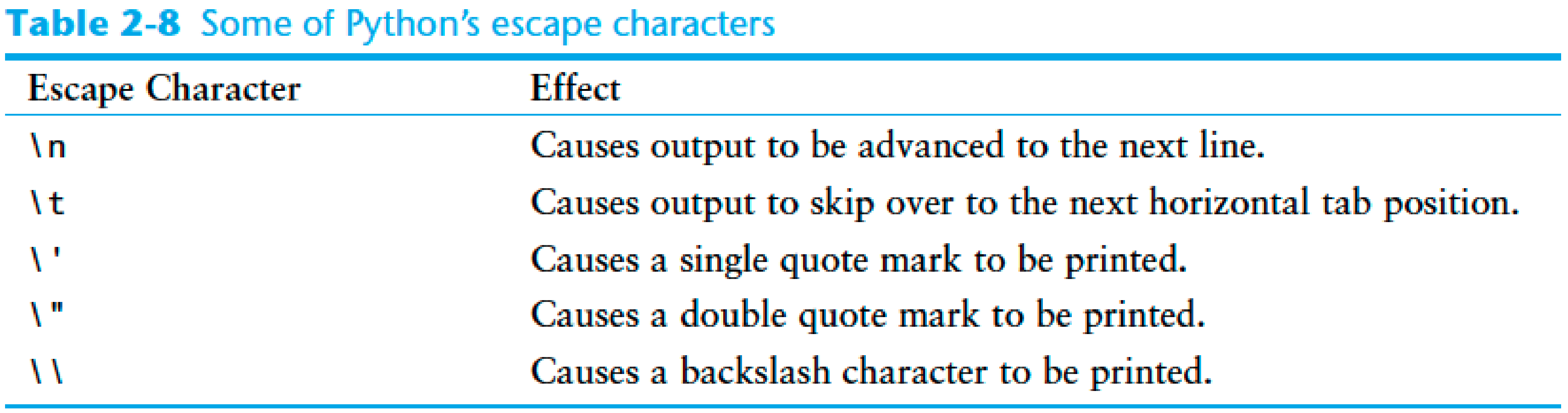


Table 2-8 Some of Python's escape characters

Escape Character	Effect
`\n`	Causes output to be advanced to the next line.
`\t`	Causes output to skip over to the next horizontal tab position.
`\'`	Causes a single quote mark to be printed.
`\"`	Causes a double quote mark to be printed.
`\\`	Causes a backslash character to be printed.

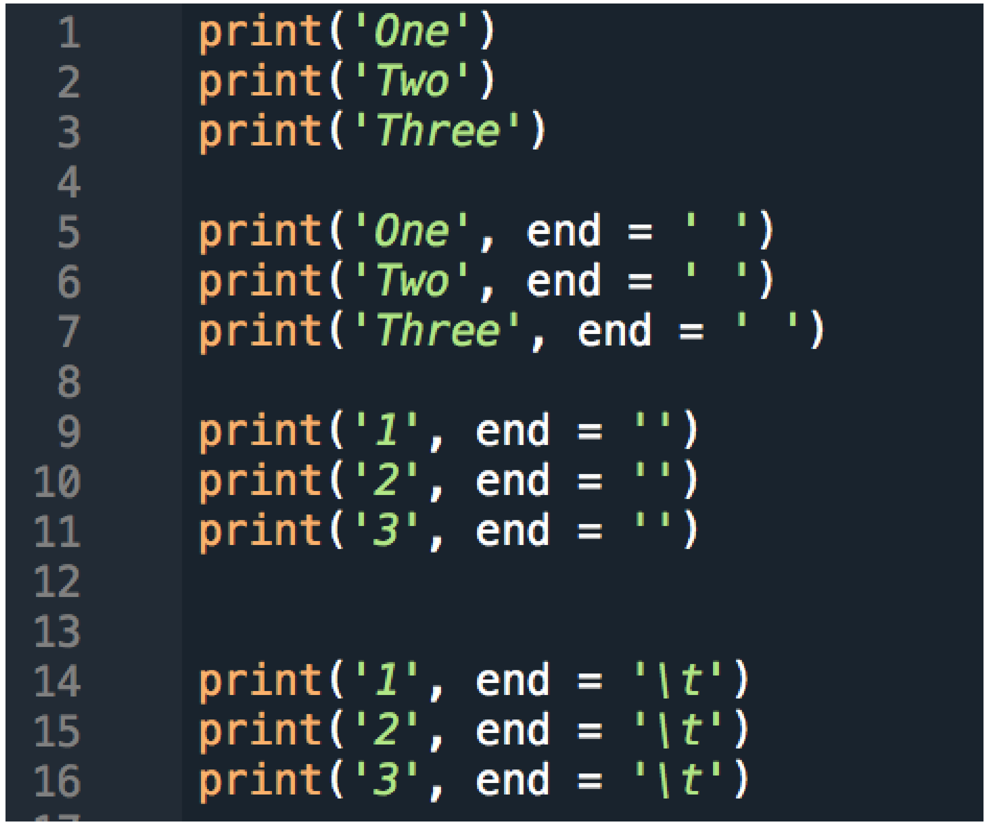
Starting out with Python, Tony Gaddis.

```
print('Finish \'Hw2\' by tomorrow.')
```

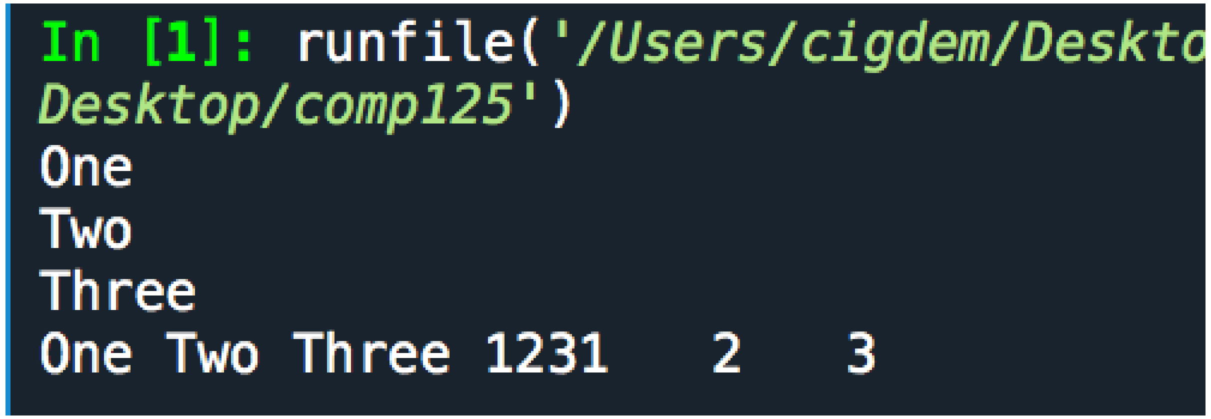
```
Finish 'Hw2' by tomorrow.
```

More on `print`

- If you do not want to end each print with a new line, you can pass the special argument **end** to the `print` function



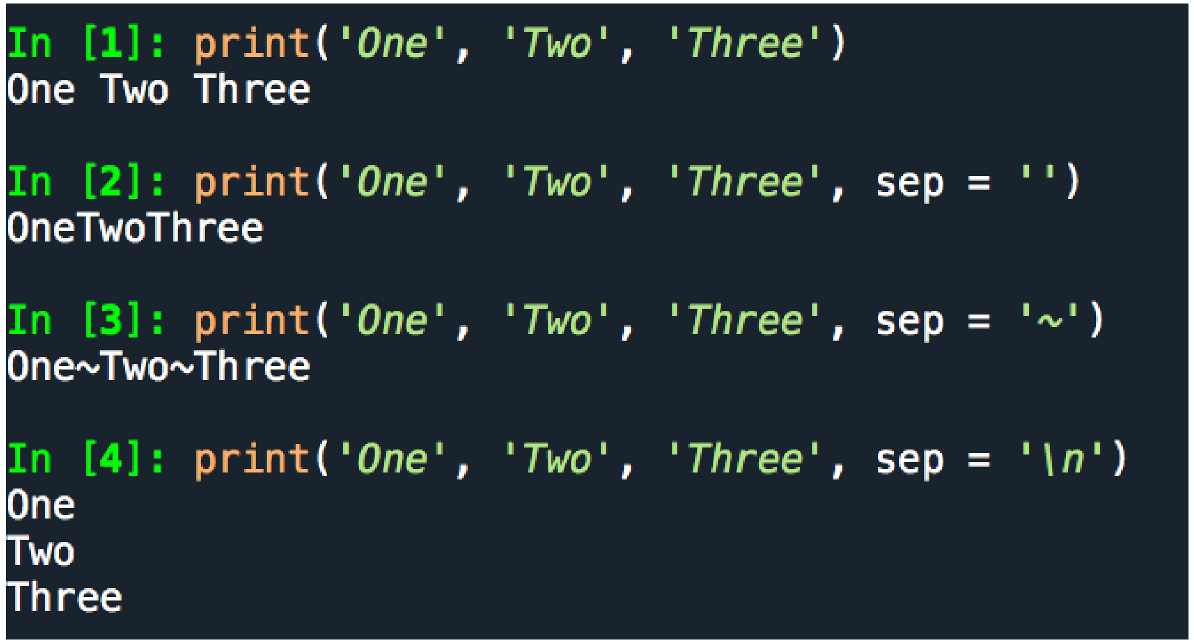
```python
print('One')
print('Two')
print('Three')

print('One', end = ' ')
print('Two', end = ' ')
print('Three', end = ' ')

print('1', end = '')
print('2', end = '')
print('3', end = '')


print('1', end = '\t')
print('2', end = '\t')
print('3', end = '\t')
```



```
In [1]: runfile('/Users/cigdem/Deskto
Desktop/comp125')
One
Two
Three
One Two Three 1231	2	3
```

More on `print`

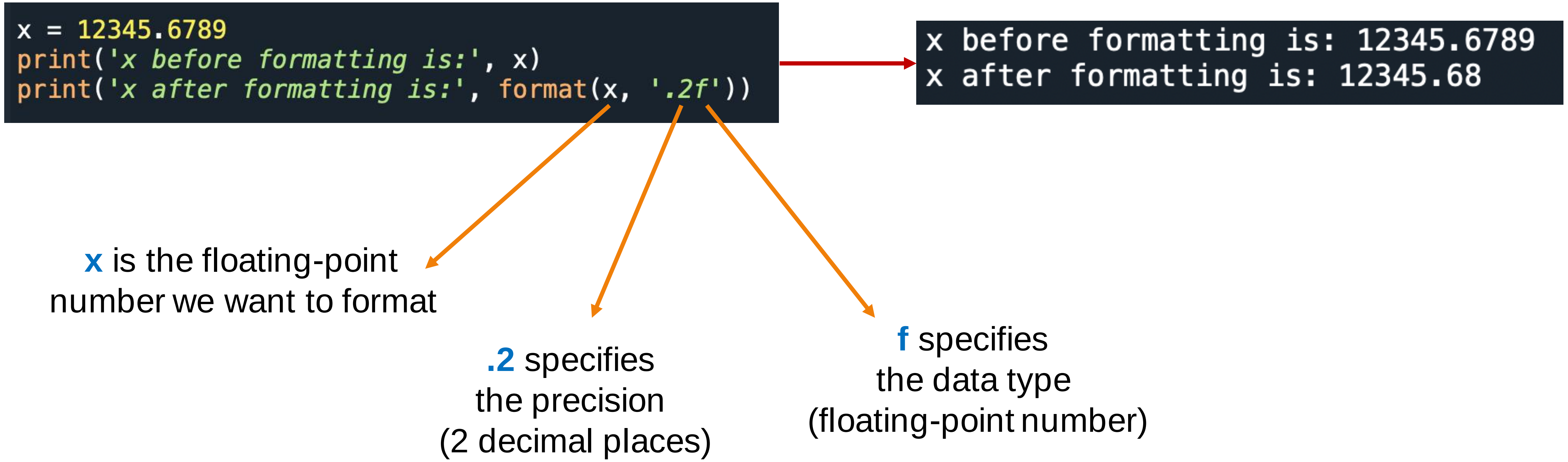
- If you pass **multiple arguments** to the print function, these arguments are **automatically separated** by a space
- If you do not want a space, you can pass the special argument **sep** to the **`print`** function, specifying the character you will use as a separator



```
In [1]: print('One', 'Two', 'Three')
One Two Three

In [2]: print('One', 'Two', 'Three', sep = '')
OneTwoThree

In [3]: print('One', 'Two', 'Three', sep = '~')
One~Two~Three

In [4]: print('One', 'Two', 'Three', sep = '\n')
One
Two
Three
```

Formatting Numbers

- When we print a floating number, it appears with up to 12 significant digits
- We can call the built-in `format` function

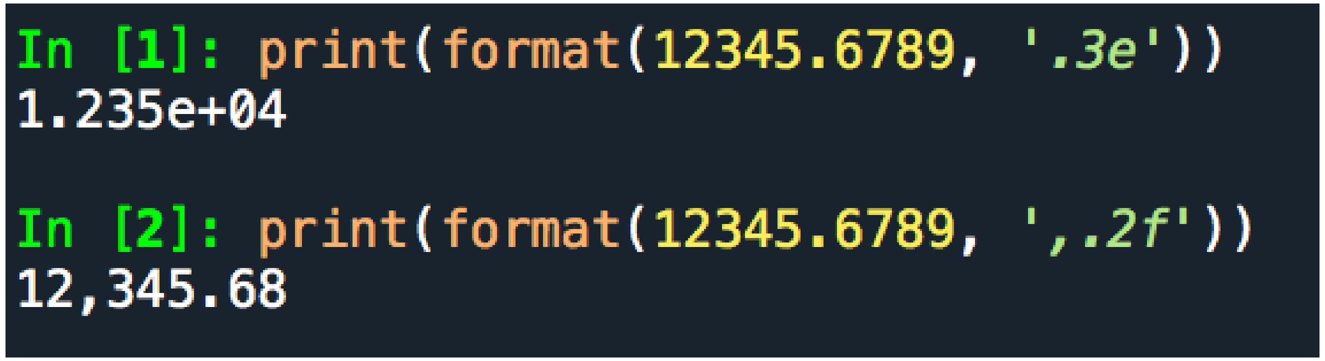


```
x = 12345.6789
print('x before formatting is:', x)
print('x after formatting is:', format(x, '.2f'))
```

```
x before formatting is: 12345.6789
x after formatting is: 12345.68
```

x is the floating-point number we want to format
.2 specifies the precision (2 decimal places)
f specifies the data type (floating-point number)

Formatting Numbers



```
In [1]: print(format(12345.6789, '.3e'))
1.235e+04

In [2]: print(format(12345.6789, ',.2f'))
12,345.68
```

Formatting in scientific notation

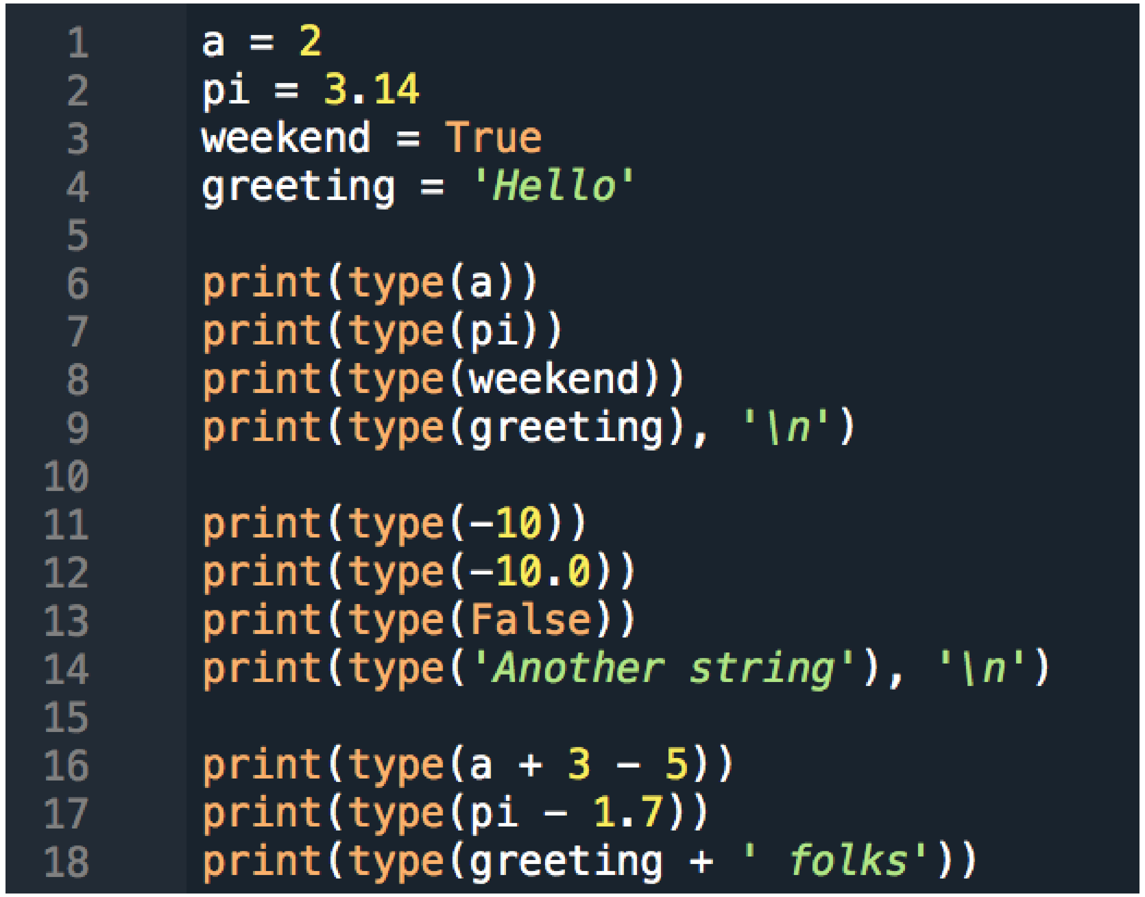
Inserting comma separators

`format(value, format_specifiers)`

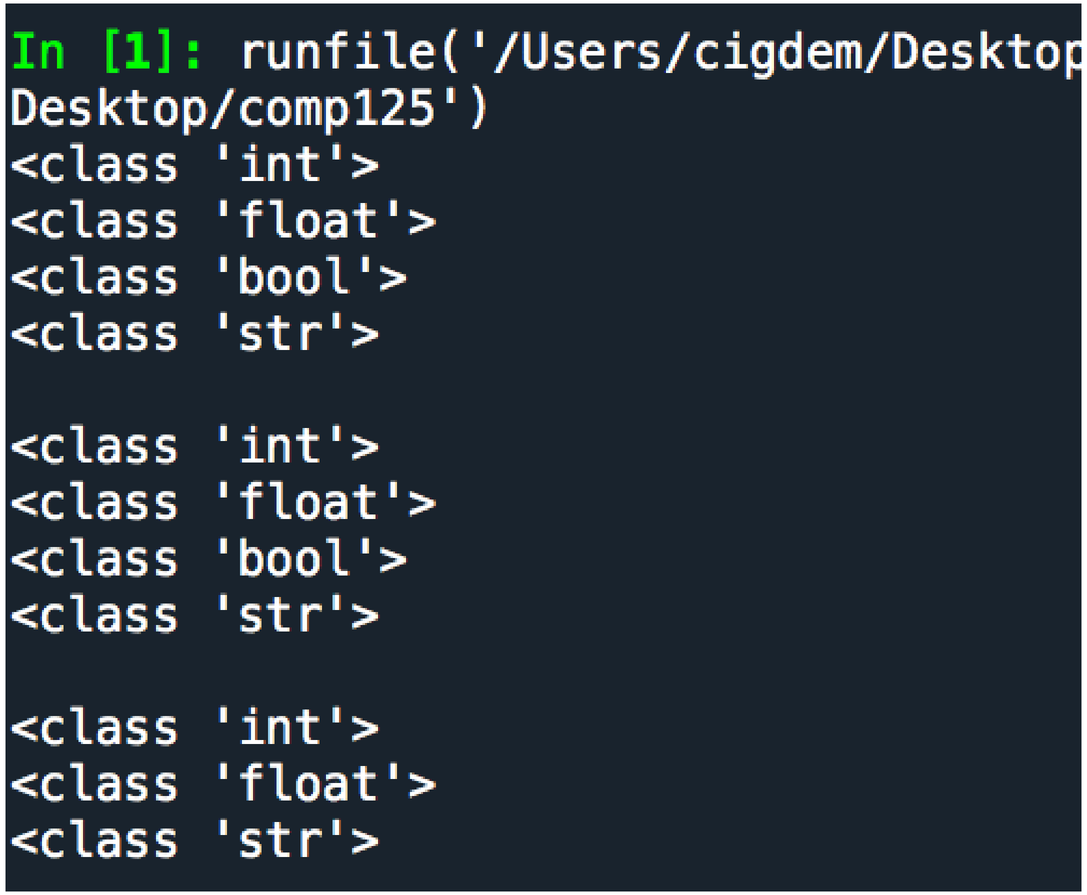
There exist other types of `format_specifiers`
For more information on its use, please refer the manual of `format`
No need to memorize the other ones

Displaying Data Types

- **`type`** function outputs the data type of a literal or a variable or an expression given as an argument



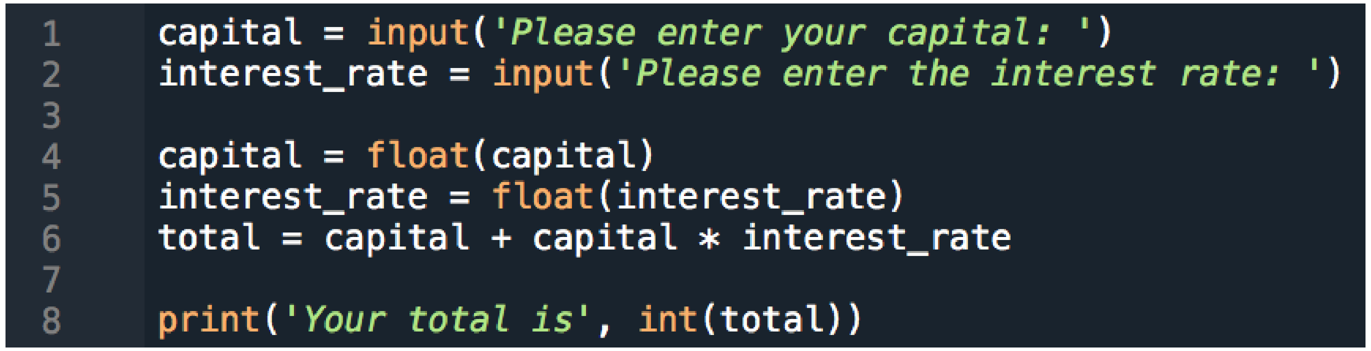
```python
a = 2
pi = 3.14
weekend = True
greeting = 'Hello'

print(type(a))
print(type(pi))
print(type(weekend))
print(type(greeting), '\n')

print(type(-10))
print(type(-10.0))
print(type(False))
print(type('Another string'), '\n')

print(type(a + 3 - 5))
print(type(pi - 1.7))
print(type(greeting + ' folks'))
```



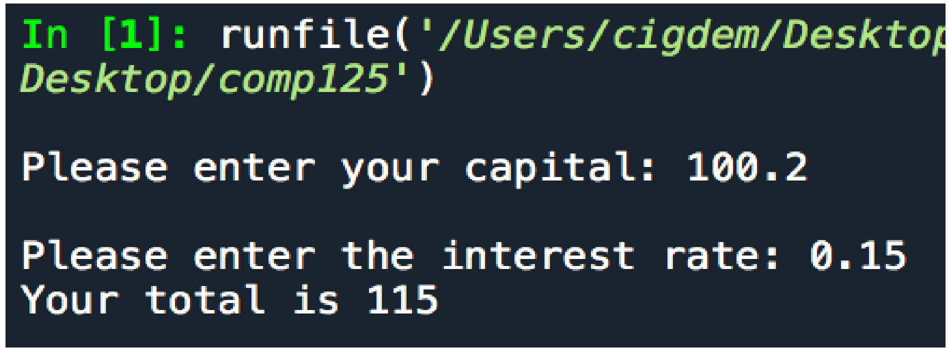
```
In [1]: runfile('/Users/cigdem/Desktop
Desktop/comp125')
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>

<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>

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

Exercise

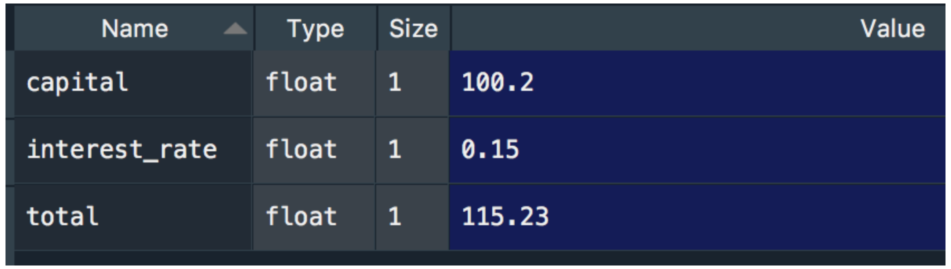
- Write a code fragment that takes a capital in dollars and an annual interest rate from a user, calculates the amount of the capital after a year, and displays the resulting amount on the screen without showing the cents part



```python
capital = input('Please enter your capital: ')
interest_rate = input('Please enter the interest rate: ')

capital = float(capital)
interest_rate = float(interest_rate)
total = capital + capital * interest_rate

print('Your total is', int(total))
```



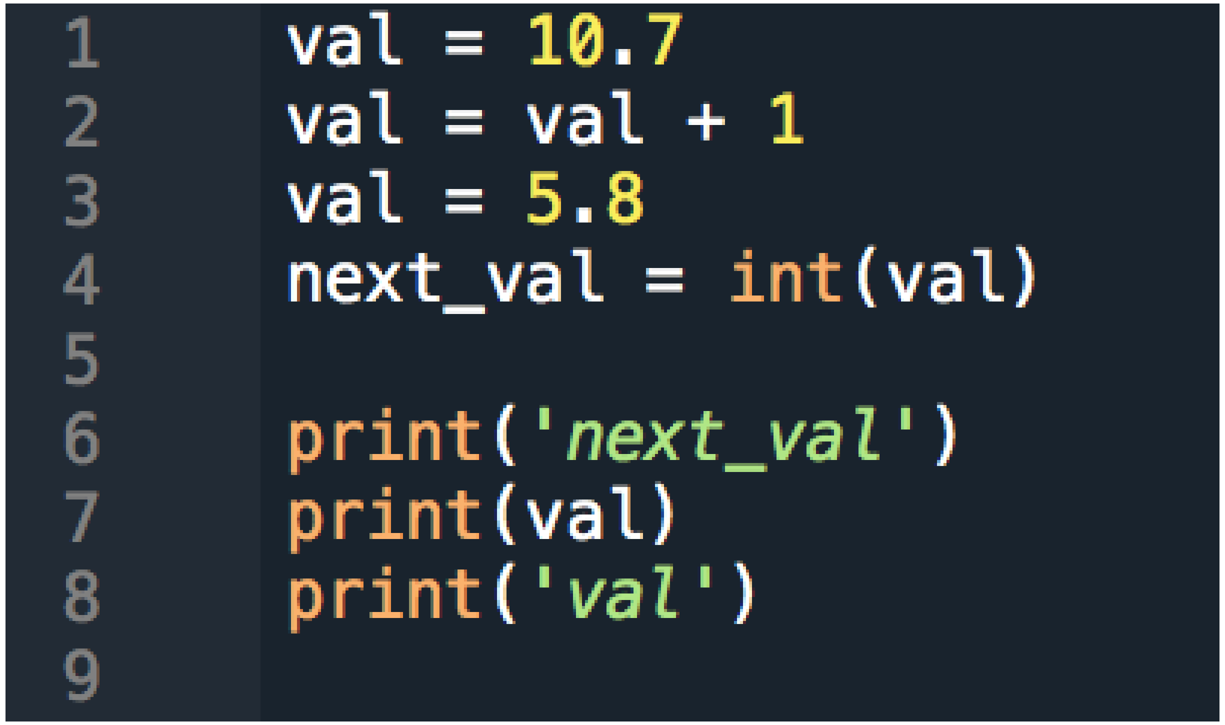
```
In [1]: runfile('/Users/cigdem/Desktop
Desktop/comp125')

Please enter your capital: 100.2

Please enter the interest rate: 0.15
Your total is 115
```



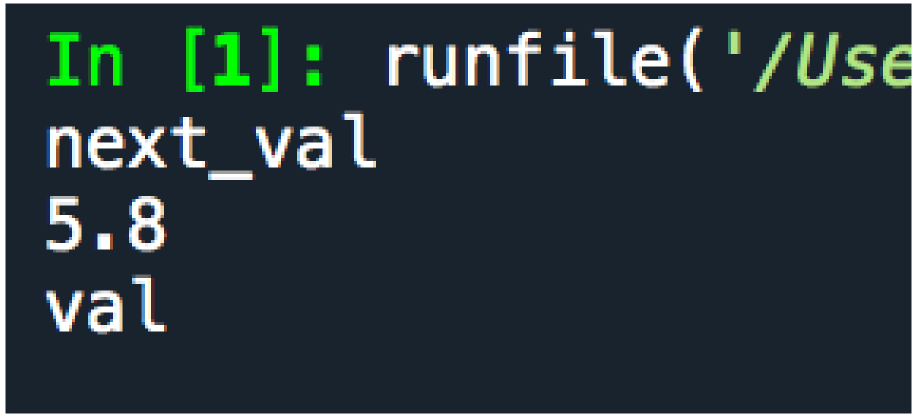
Name	Type	Size	Value
capital	float	1	100.2
interest_rate	float	1	0.15
total	float	1	115.23

Exercise

- What is the output of the following code fragment?



```python
val = 10.7
val = val + 1
val = 5.8
next_val = int(val)

print('next_val')
print(val)
print('val')

```



```
In [1]: runfile('/Use
next_val
5.8
val
```

Expressions, Statements, Comments

Expressions and Statements

- An **expression** is a combination of literal constants (or simply literals), variables, and operators
- A **statement** is a unit of code that has an effect, like creating a variable or displaying a value

```
In [3]: 42                expression
Out[3]: 42

In [4]: n = 42            statement

In [5]: n + 25            expression
Out[5]: 67

In [6]: z = n + 25        statement

In [7]: z                 expression
Out[7]: 67

In [8]: print(z)          statement
67
```

Breaking Long Statements

- You can use *line continuation character*, which is a backslash (\), to break a statement into multiple lines
  - *e.g.,*

```
result = var1 * 2 + var2 * 3 + \
         var3 * 4 + var4 * 5
```

- You don’t need a line continuation character to break expressions **enclosed in parentheses** into multiple lines.
  - *e.g.,*

```
total = (value1 + value2 +
         value3 + value4 +
         value5 + value6)
```

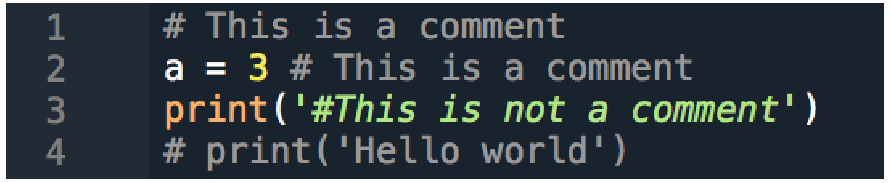
Combining Multiple Statements

- You can use *semi-colon character (;)* to chain multiple statements in a single line
  - *e.g.,*

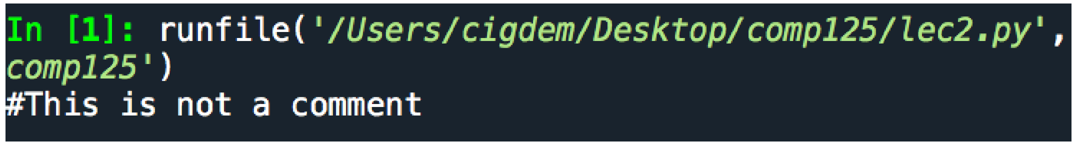
```
x = 5 ; y = 3.4 ; a = x + y ; print(a)
```

Commenting

- A comment is a line(s) of text in a program that will not be executed by the interpreter
- Comments are to increase the readability for humans
- Single-line comments:
  - The text followed by the # character
  - Can start anywhere in a line
- Multi-line comments:
  - Use the # character at each line
  - Add a multiline string (triple quotes)



```python
# This is a comment
a = 3 # This is a comment
print('#This is not a comment')
# print('Hello world')
```



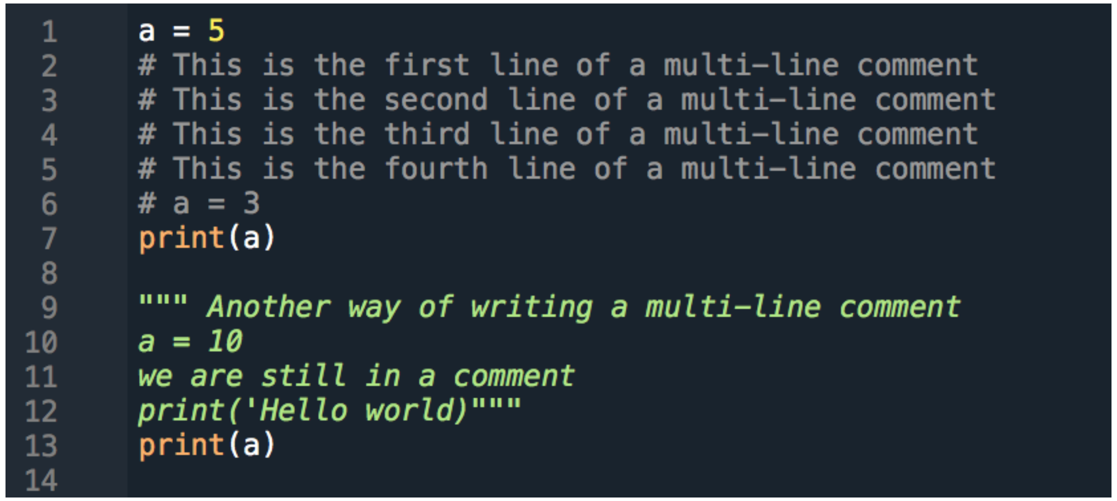
```
In [1]: runfile('/Users/cigdem/Desktop/comp125/lec2.py',
comp125')
#This is not a comment
```

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\textit{Single-line comments}}$

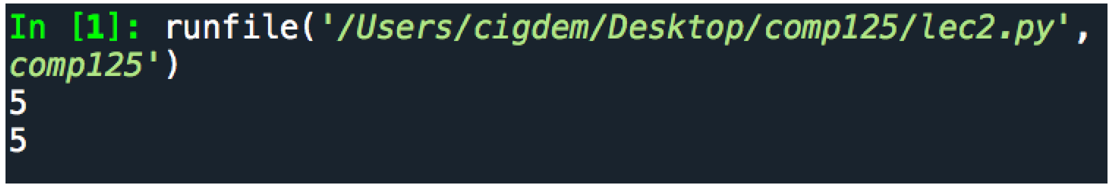
Commenting is very important especially when you are working in a group of people. **Thus, we will enforce you to write explanatory and clearly understandable comments in your homework assignments, which will affect your grade!!!**

Commenting

- Multi-line comments:
  - Use the # character at each line
  - Add a multiline string (triple single-quotes (`'''`) or double-quotes (`"""`))



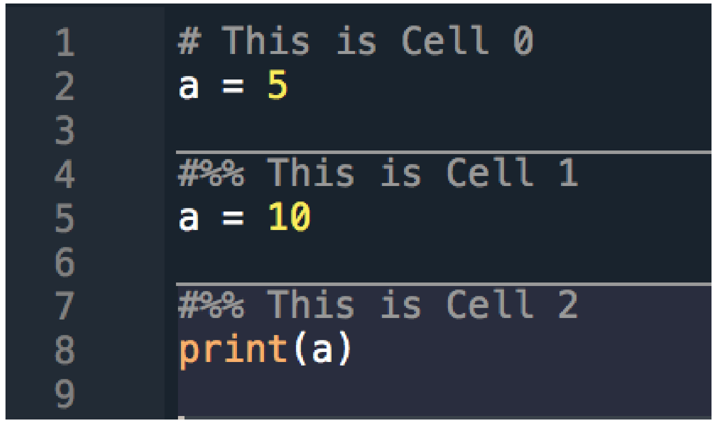
```
a = 5
# This is the first line of a multi-line comment
# This is the second line of a multi-line comment
# This is the third line of a multi-line comment
# This is the fourth line of a multi-line comment
# a = 3
print(a)

""" Another way of writing a multi-line comment
a = 10
we are still in a comment
print('Hello world)"""
print(a)

```



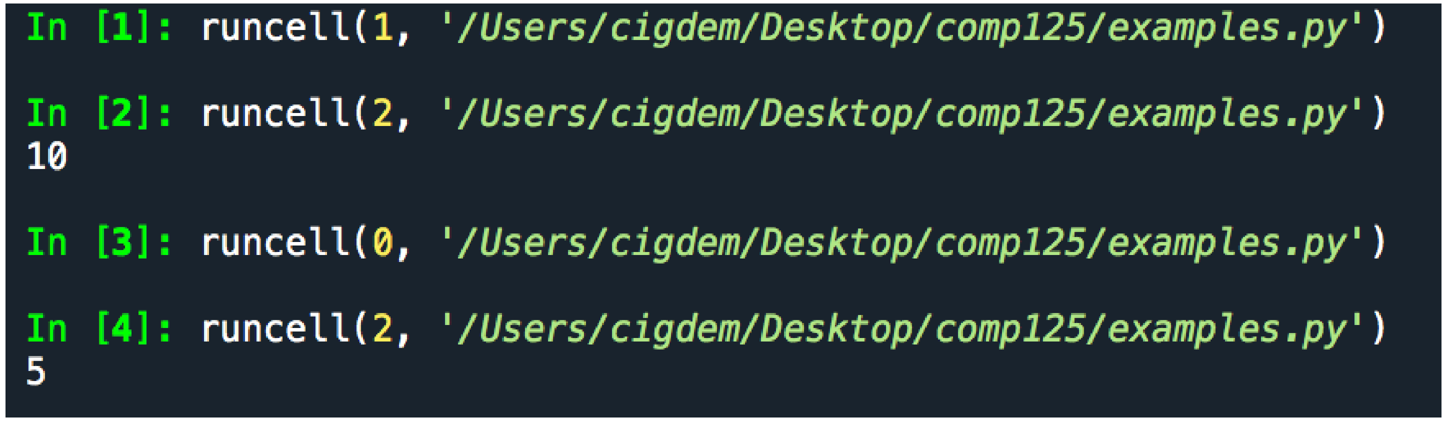
```
In [1]: runfile('/Users/cigdem/Desktop/comp125/lec2.py',
comp125')
5
5
```

Separating Code into Individual Cells

- You can use `#%%` to separate your code into individual cells
- Each cell can be individually and independently called
- Cells are labeled starting with 0 (not 1)



```python
# This is Cell 0
a = 5

#%% This is Cell 1
a = 10

#%% This is Cell 2
print(a)
```



```
In [1]: runcell(1, '/Users/cigdem/Desktop/comp125/examples.py')

In [2]: runcell(2, '/Users/cigdem/Desktop/comp125/examples.py')
10

In [3]: runcell(0, '/Users/cigdem/Desktop/comp125/examples.py')

In [4]: runcell(2, '/Users/cigdem/Desktop/comp125/examples.py')
5
```

Arithmetic Operations

Performing Calculations

- We use **math expressions** to perform calculations
- Python provides several **math operators** to create these math expressions

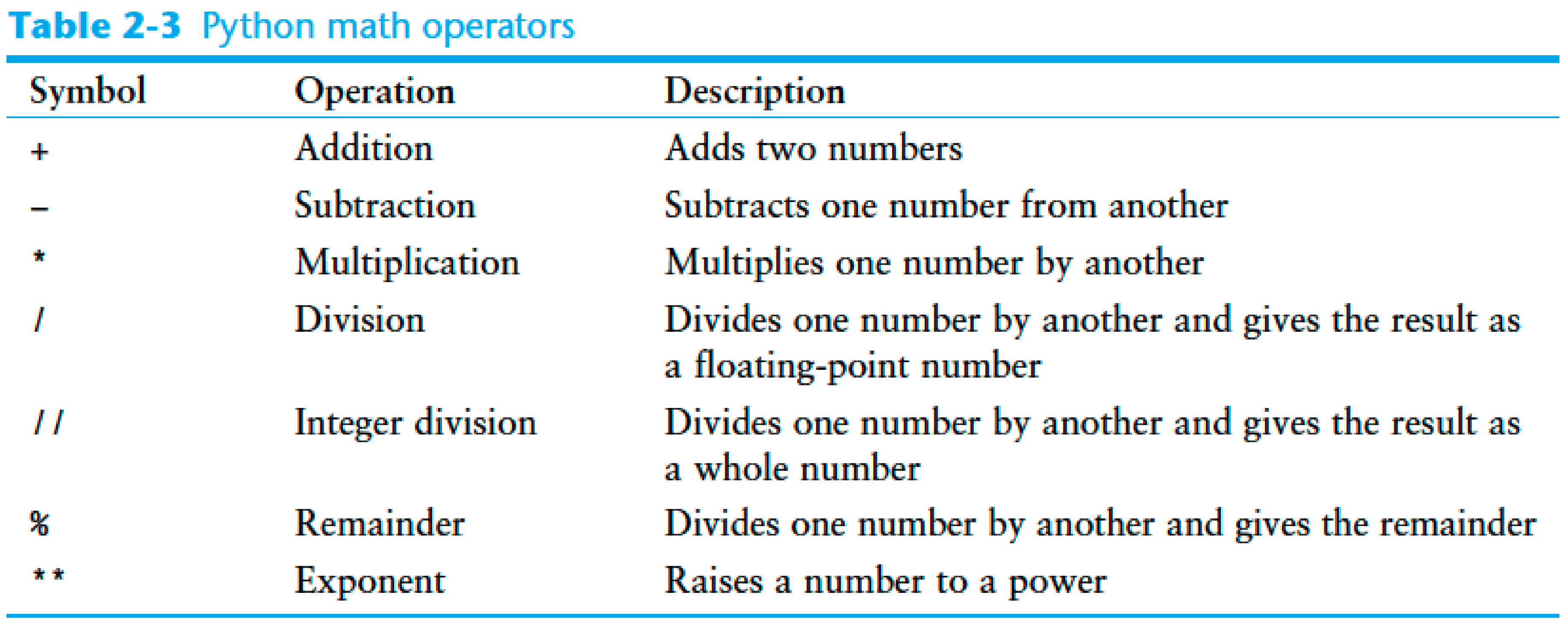


Table 2-3 Python math operators

Symbol	Operation	Description
+	Addition	Adds two numbers
–	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the result as a floating-point number
//	Integer division	Divides one number by another and gives the result as a whole number
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

Starting out with Python, Tony Gaddis.

Operator Precedence

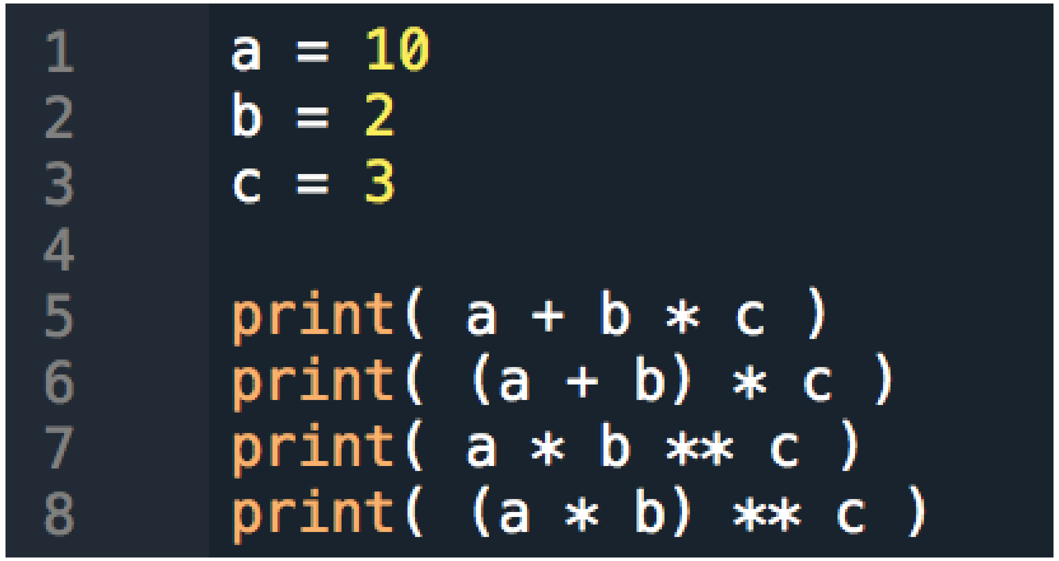
- Python follows the rules you learned in your math classes

- Operators from the highest to lowest precedence are as follows:
  - Parentheses (highest precedence; operations enclosed in parentheses are performed first)
  - Exponentiation: **
  - Multiplication, division and remainder: * , / , // , %
  - Addition and subtraction: + , -

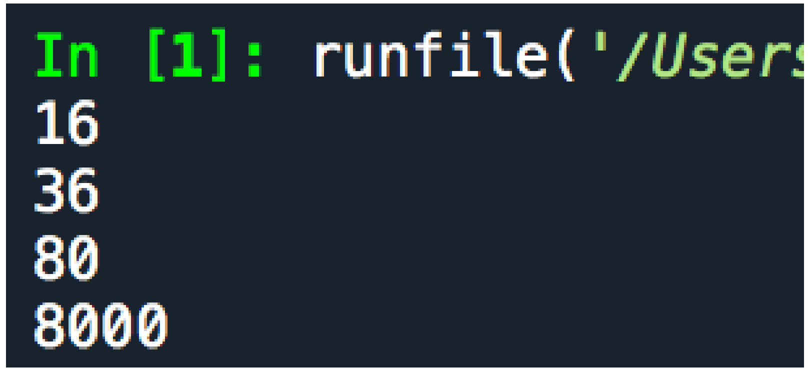
- When operators have the same precedence, they are evaluated from left-to-right
  - 12 / 2 * 3 → 18 [**NOT 2**, which is the result of 12 / (2 * 3)]
  - **Exception:** ** is evaluated from right-to-left
  - 2 ** 3 ** 2 → 512 [**NOT 64**, which is the result of (2 ** 3) ** 2]

Exercise

- What is the output of the following code fragment?



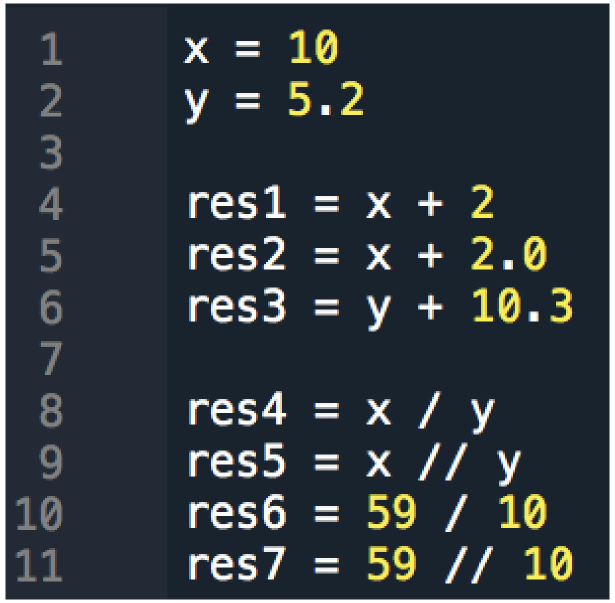
```
a = 10
b = 2
c = 3

print( a + b * c )
print( (a + b) * c )
print( a * b ** c )
print( (a * b) ** c )
```



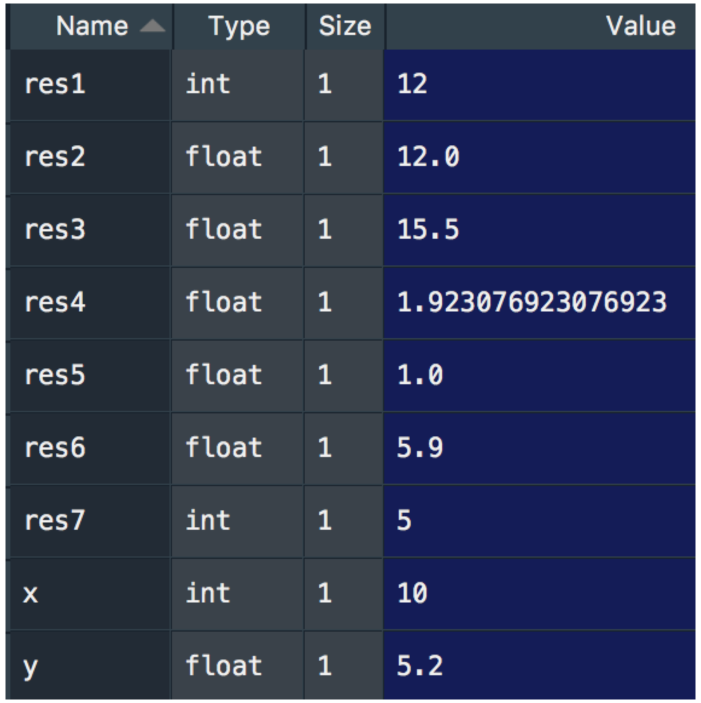
```
In [1]: runfile('/Users
16
36
80
8000
```

Mixed-Type Expressions on Numeric Data Types

- An operation performed on two **int** operands gives an **int** result
  - Starting from Python version 3, division of two **int** values gives a **float** result
- An operation performed on two **float** operands gives a **float** result
- An operation performed on an **int** and a **float** operand gives a **float** result
- An integer division // always gives an **int** result



```
x = 10
y = 5.2

res1 = x + 2
res2 = x + 2.0
res3 = y + 10.3

res4 = x / y
res5 = x // y
res6 = 59 / 10
res7 = 59 // 10
```

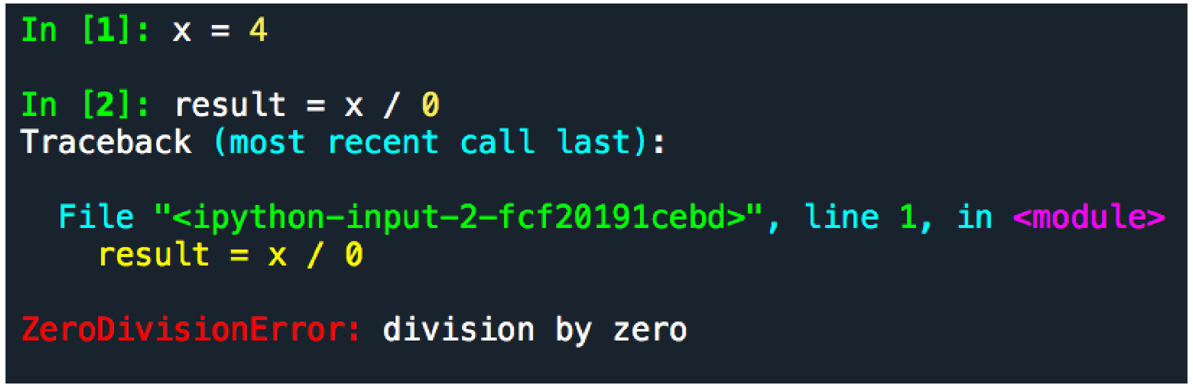


Name	Type	Size	Value
res1	int	1	12
res2	float	1	12.0
res3	float	1	15.5
res4	float	1	1.923076923076923
res5	float	1	1.0
res6	float	1	5.9
res7	int	1	5
x	int	1	10
y	float	1	5.2

Exercise

We have w = 89, x = 4, y = 8, and z = 10
What value will be stored in the *result* variable after each arithmetic operation?

`result = x + y`	12
`result = z * 2`	20
`result = y / x`	2.0
`result = y // x`	2
`result = y - z`	-2
`result = w // z`	8
`result = w / z`	8.9
`result = w % z`	9
`result = z ** x`	10000
`result = x + 0`	4
`result = x / 0`	run-time error !!!



```
In [1]: x = 4

In [2]: result = x / 0
Traceback (most recent call last):

  File "<ipython-input-2-fcf20191cebd>", line 1, in <module>
    result = x / 0

ZeroDivisionError: division by zero
```

Exercise

- Convert the following math formulas to programming statements

  Do not use parentheses unless they are necessary

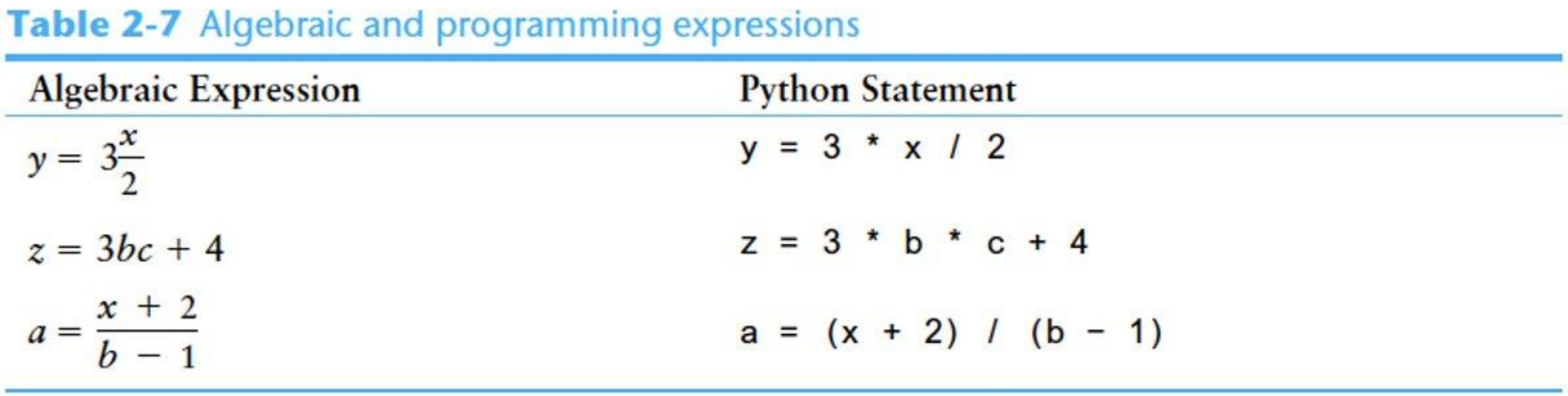


Table 2-7 Algebraic and programming expressions

Algebraic Expression	Python Statement
$y = 3\frac{x}{2}$	`y = 3 * x / 2`
$z = 3bc + 4$	`z = 3 * b * c + 4`
$a = \frac{x + 2}{b - 1}$	`a = (x + 2) / (b - 1)`

Starting out with Python, Tony Gaddis.

Strings Operations

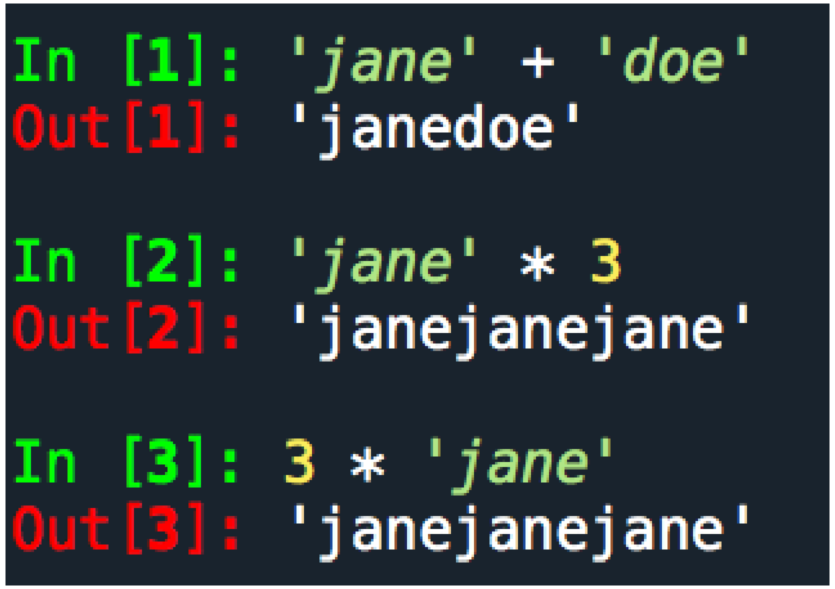
- + is defined for two **str** operands

  + performs string concatenation

- * is defined for one **str** and one **int** operand

  ** performs string repetition*

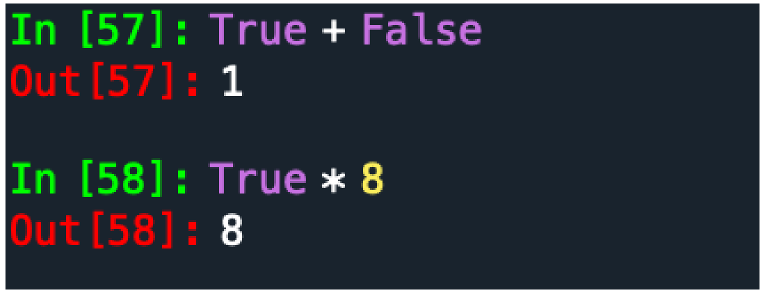
- Other operations are invalid
  - + is invalid if one operand is **str** but the other is not
  - * is invalid unless one operand is **str** and the other is **int**
  - All other math operations are invalid



```
In [1]: 'jane' + 'doe'
Out[1]: 'janedoe'

In [2]: 'jane' * 3
Out[2]: 'janejanejane'

In [3]: 3 * 'jane'
Out[3]: 'janejanejane'
```

Boolean Operations

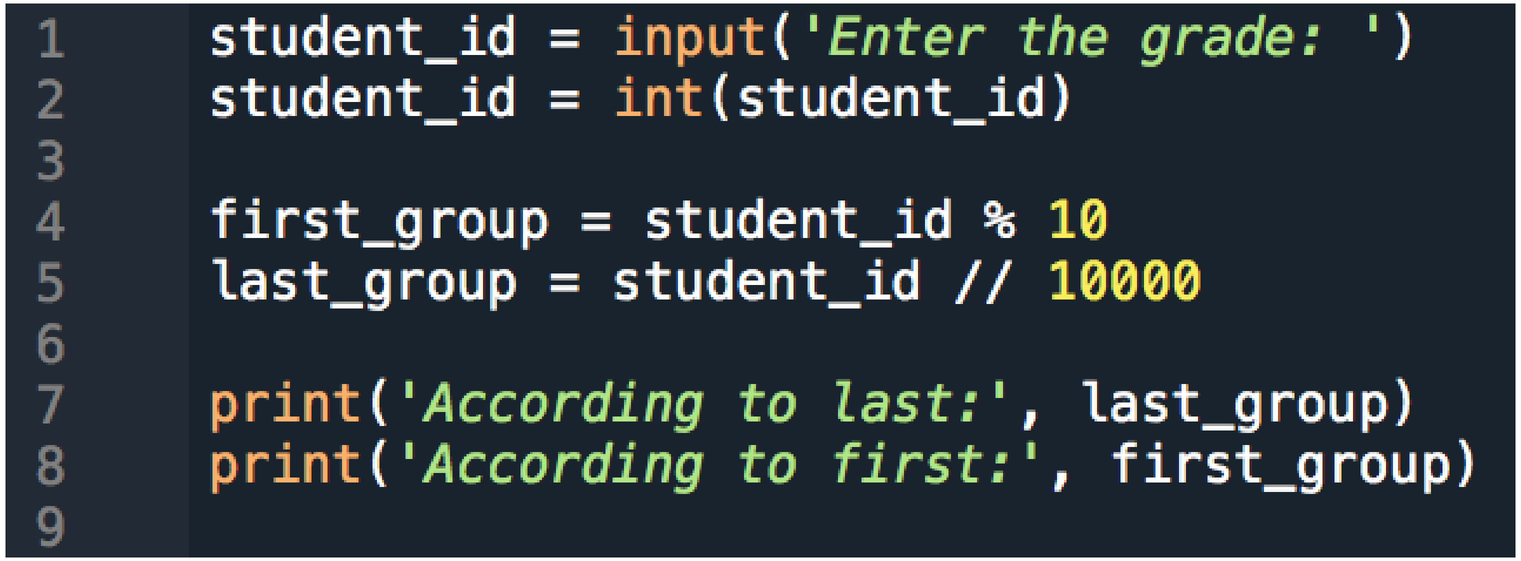
- Python treats bool values in arithmetic expressions as integers.
  - True = 1, False = 0



```
In [57]: True + False
Out[57]: 1

In [58]: True * 8
Out[58]: 8
```

Exercise

- Suppose that we want to group students into 10 (groups from 0 to 9), first according to their last digits and then according to their first digits
- Write a code fragment that takes the id of a student and displays the two groups that s/he belongs to
- You may assume that a student id is a 5-digit integer



```python
student_id = input('Enter the grade: ')
student_id = int(student_id)

first_group = student_id % 10
last_group = student_id // 10000

print('According to last:', last_group)
print('According to first:', first_group)

```

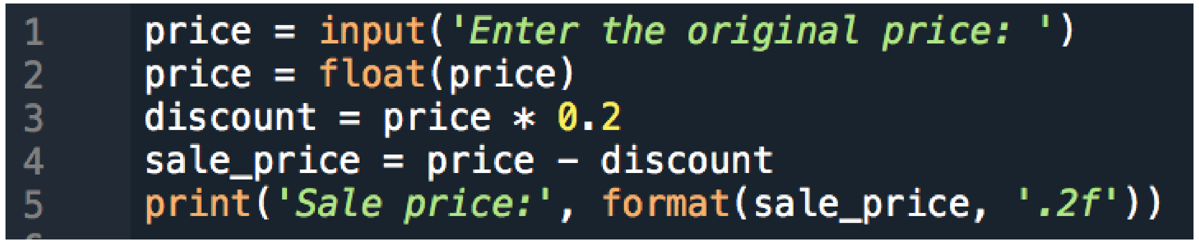
More Exercises

Exercise

- A retail business is planning to have a storewide sale where the prices of all items will be **20%** off. Write a program to calculate the **sale price** of an item **after the discount** is subtracted.
  - First write its pseudocode
  - Then implement the algorithm in Python

Pseudocode

```
Get the original price of an item from the user

Calculate 20 percent of the original price (this is the amount of discount)

Subtract the discount from the original price (this is the sale price)

Display the sale price
```



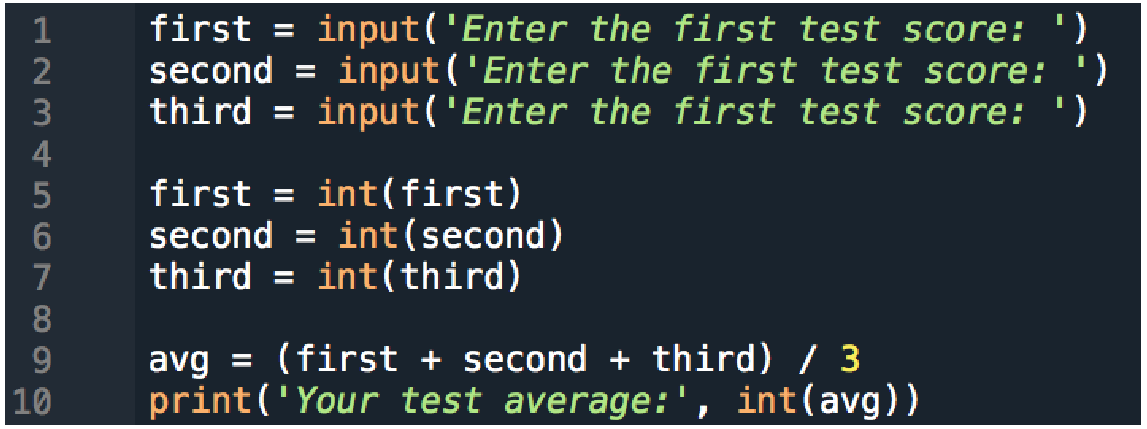
```python
price = input('Enter the original price: ')
price = float(price)
discount = price * 0.2
sale_price = price - discount
print('Sale price:', format(sale_price, '.2f'))
```

Exercise

- Suppose you want to learn the average of your three test scores, all of which are announced as integers. Write a program that takes the scores of these three tests, calculates their average, and displays only its integer part (ignore the values after the decimal point).
  - First write its pseudocode
  - Then implement the algorithm in Python

Pseudocode

```
Get the first test score

Get the second test score

Get the third test score

Calculate their average (their sum divided by 3)

Display the average in the required format
```



```python
first = input('Enter the first test score: ')
second = input('Enter the first test score: ')
third = input('Enter the first test score: ')

first = int(first)
second = int(second)
third = int(third)

avg = (first + second + third) / 3
print('Your test average:', int(avg))
```

Exercise

- Suppose you want to deposit a certain amount of money into your savings account and leave it alone to draw interest for the next 10 years. At the end of the 10 years, you want to have \$10,000 in your account. How much do you need to deposit today to make that happen when an interest rate is given?
  - First write its pseudocode
  - Then implement the algorithm in Python

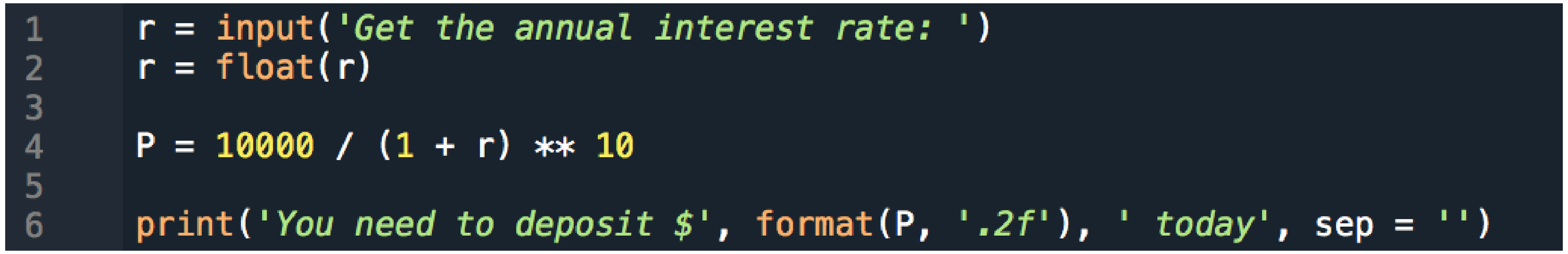
You may use this given formula

$$P = \frac{F}{(1+r)^n}$$

P : present value
F : future value (in our case \$10,000)
r : annual interest rate
n : number of years (in our case 10 years)

Pseudocode

```
Get the annual interest rate r

Calculate the amount P that will have to be deposited

Display the result of this calculation (the value of P)
```



```python
r = input('Get the annual interest rate: ')
r = float(r)

P = 10000 / (1 + r) ** 10

print('You need to deposit $', format(P, '.2f'), ' today', sep = '')
```

Simple Questions

- What does the statement **`print(format(65.5351, '.2f'))`** display?
- What does the statement **`print(format(65.5351, '.0f'))`** display?
- What does the statement **`print(int(65.5351))`** display?
- Which of the following statements will cause an error?

```
(a)   x = 17
(b)   17 = x
(c)   x = '17'
```

Simple Questions

- Which built-in function is used to convert an integer to a decimal point number?
  - (a) `int_to_float()`
  - (b) `float()`
  - (c) `convert()`
  - (d) `int()`

- Which built-in function is used to read input that was typed on the keyboard?
  - (a) `input()`
  - (b) `get_input()`
  - (c) `read_input()`
  - (d) `keyboard()`

True or False?

1. Programmers must be careful not to make syntax errors when writing pseudocodes
   FALSE
2. Variable names can have spaces in them
   FALSE
3. In Python, the first character of a variable name <u>cannot be a number</u>
   TRUE, must be one of the letters (*a-z* or *A-Z*) or an underscore character (_)
4. When you print a variable that has not been assigned a value, 0 will be displayed
   FALSE, it will raise an error