INFO-664-01 Programming For Cultural Heritage

Intro to Python:
Python Basics

Agenda

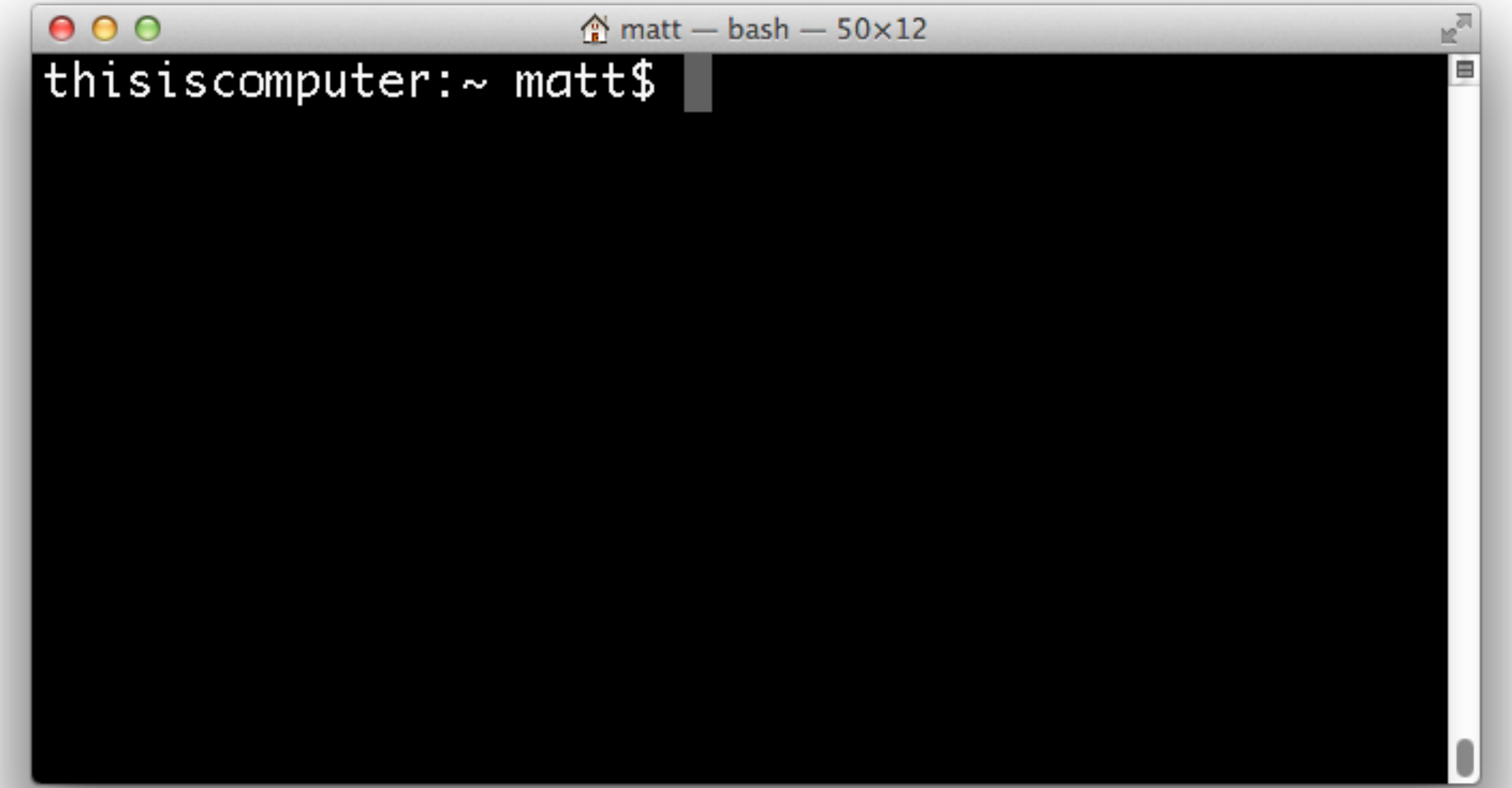
- About Python
- How do I Python
- Grammer
- Hello World
- Variables
- Control Structures
- Functions and Methods
- Modules
- Challenge

Python

- High level scripting language.
- Feed the interpreter a script and it executes synchronously.
- Is object oriented, but there are multiple ways to do the same thing.
- The script files normally have the file extension *.py*

Python Interpreter

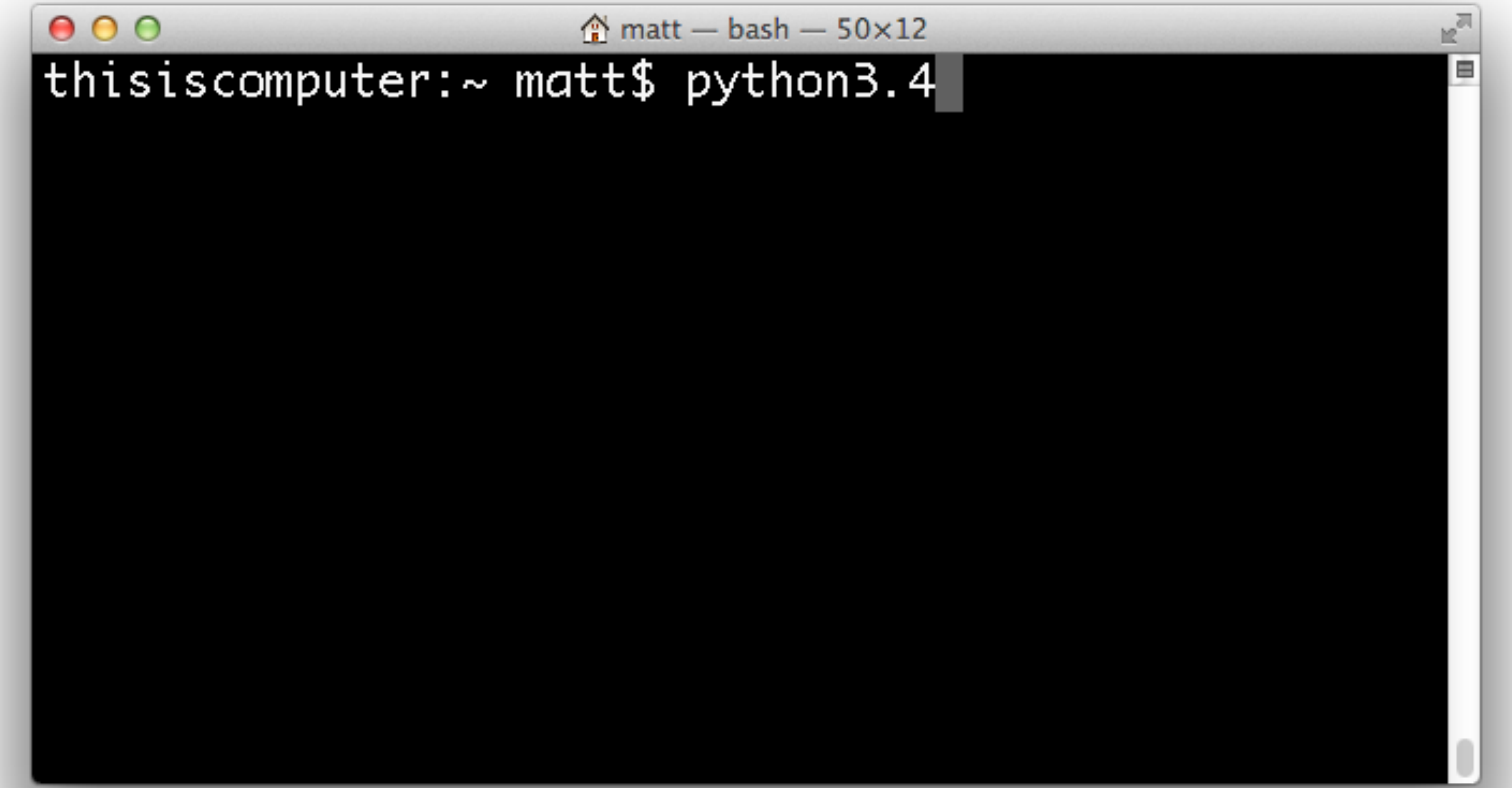
- Interactive Mode



```
thisiscomputer:~ matt$
```

Python Interpreter

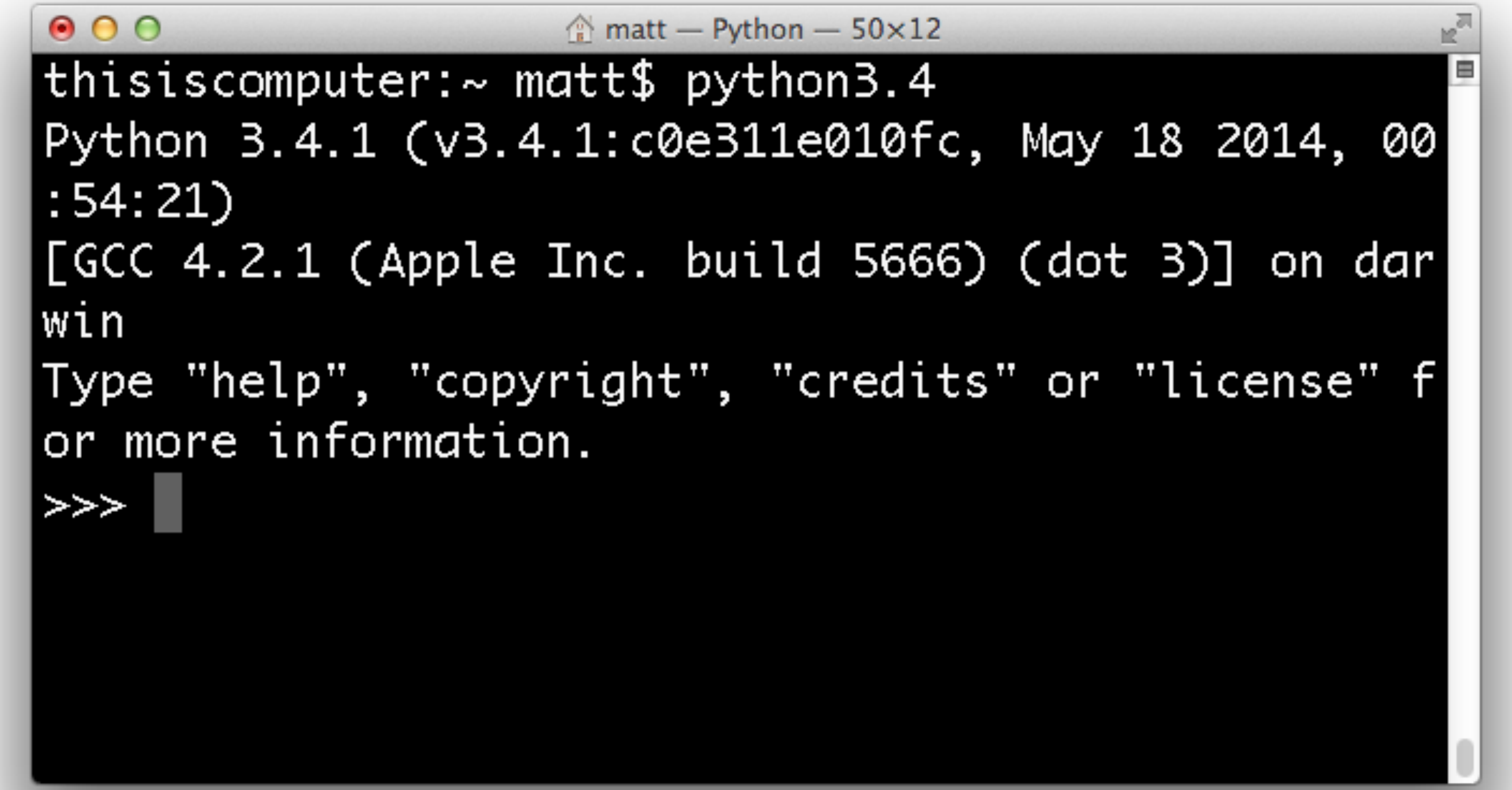
- Interactive Mode



```
thisiscomputer:~ matt$ python3.4
```

Python Interpreter

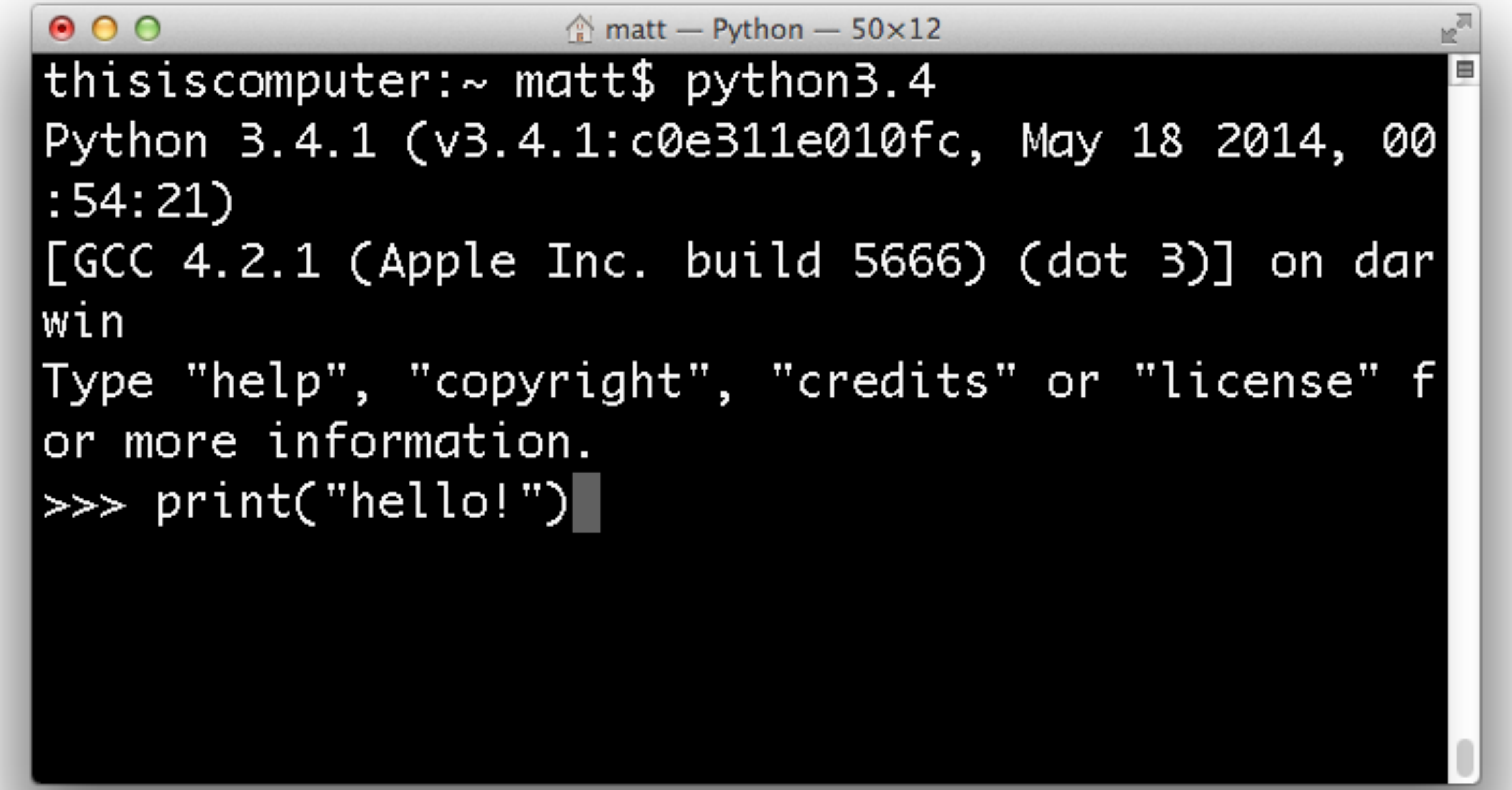
- Interactive Mode



```
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00
:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on dar
win
Type "help", "copyright", "credits" or "license" f
or more information.
>>> 
```

Python Interpreter

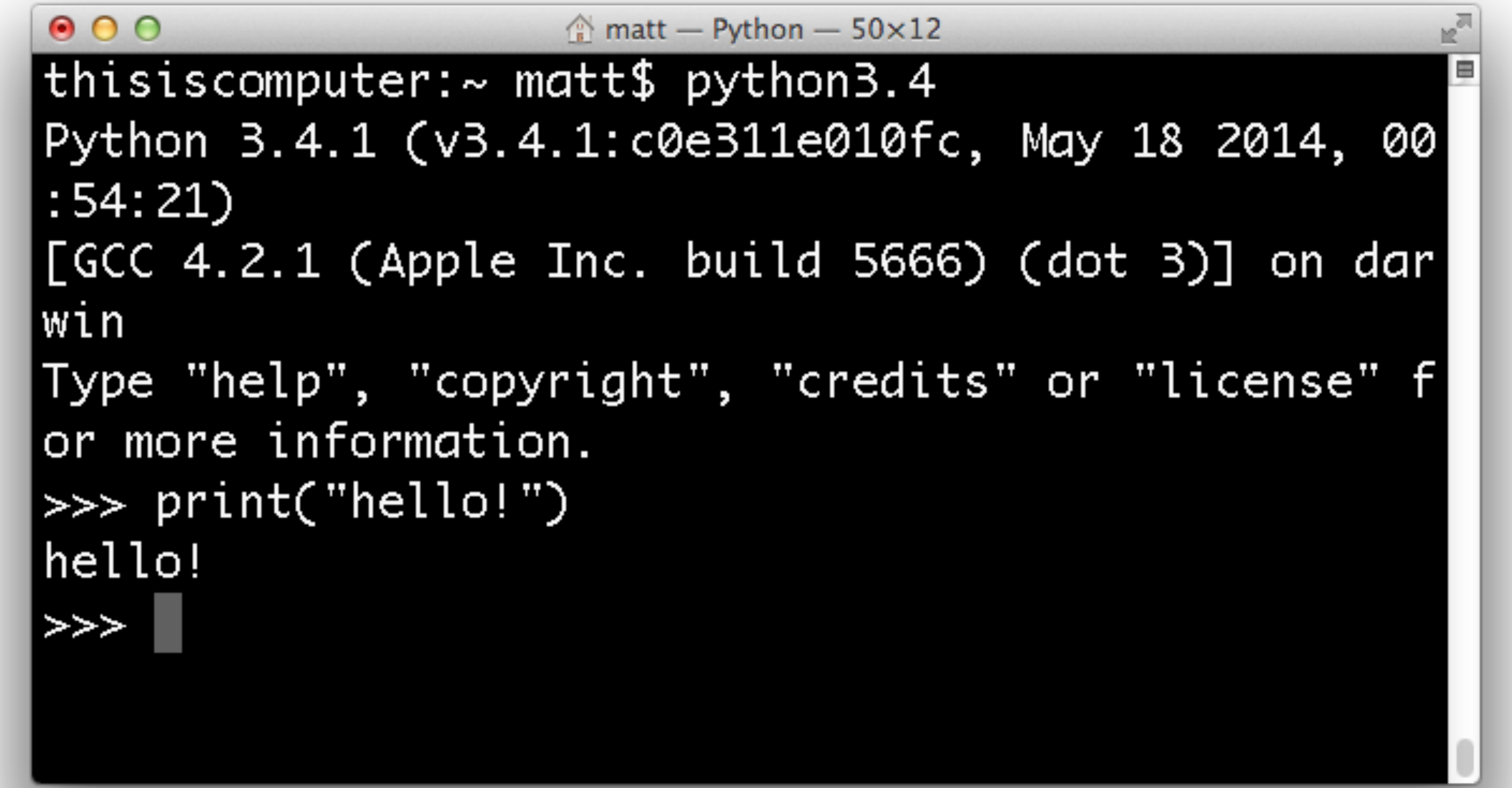
- Interactive Mode



```
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00
:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on dar
win
Type "help", "copyright", "credits" or "license" f
or more information.
>>> print("hello!")
```

Python Interpreter

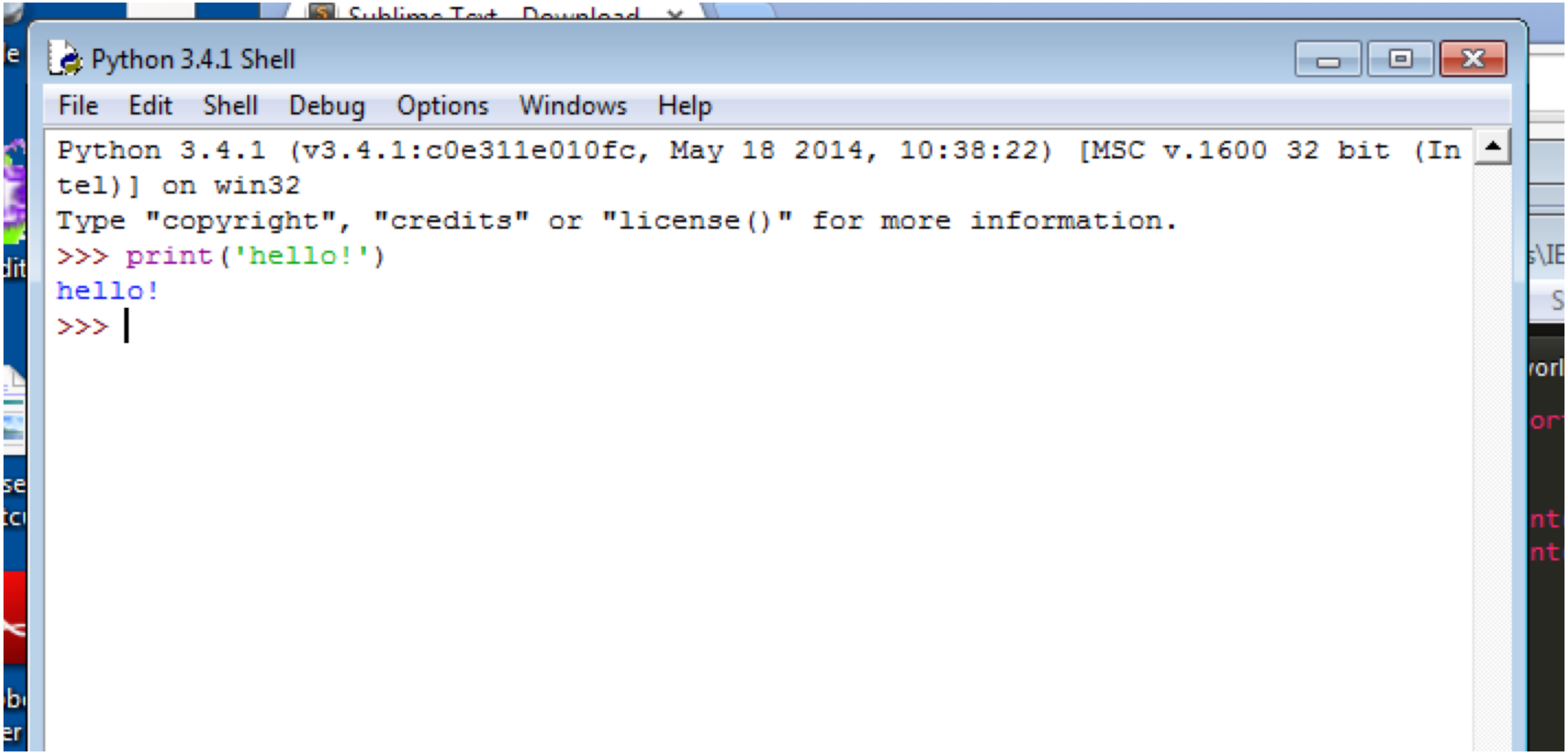
- Interactive Mode (control+D to exit)



```
thisiscomputer:~ matt$ python3.4
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00
:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on dar
win
Type "help", "copyright", "credits" or "license" f
or more information.
>>> print("hello!")
hello!
>>> 
```

Python Interpreter

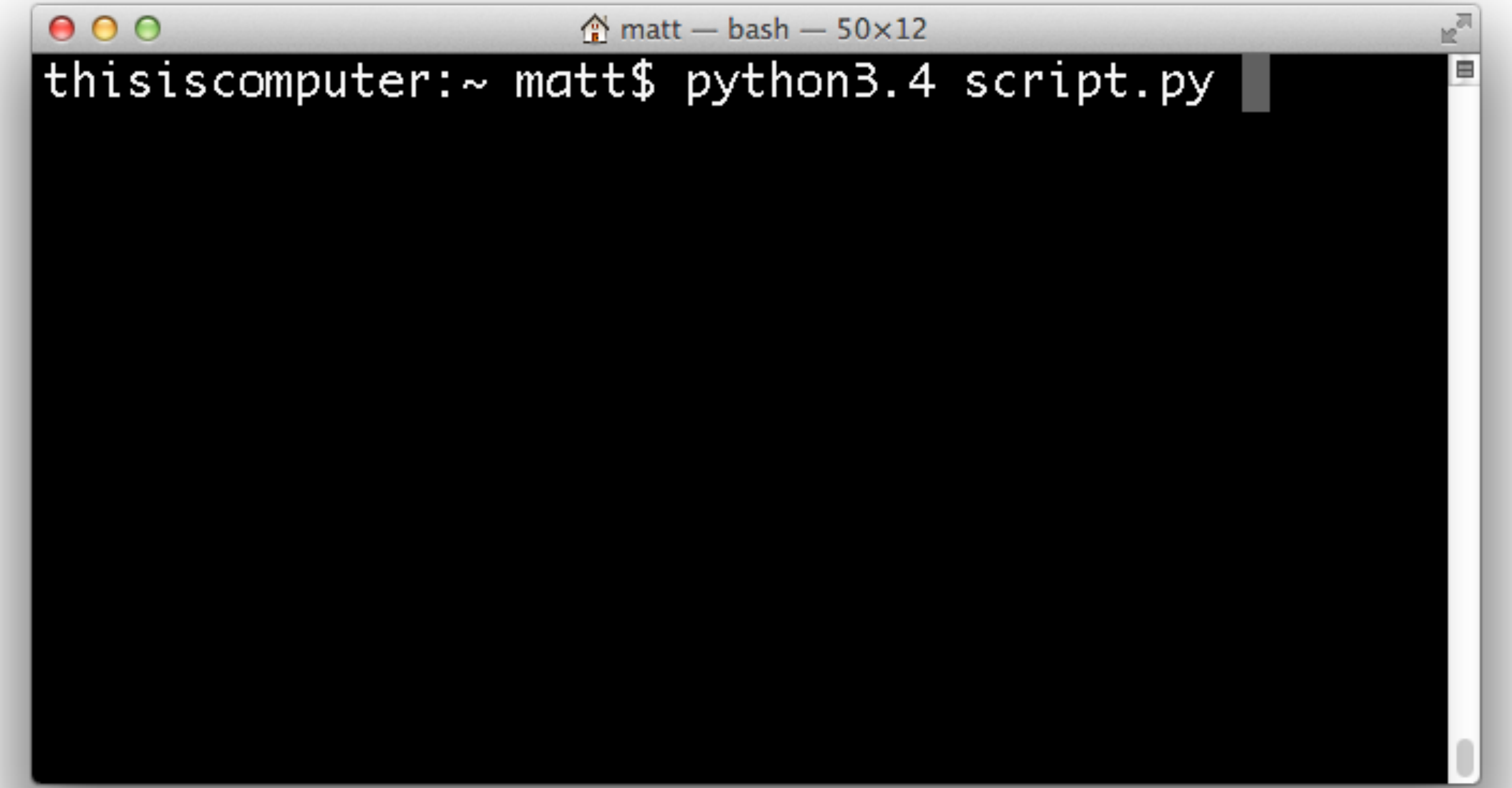
- Interactive Mode (Windows)



```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('hello!')
hello!
>>> 
```

Python Interpreter

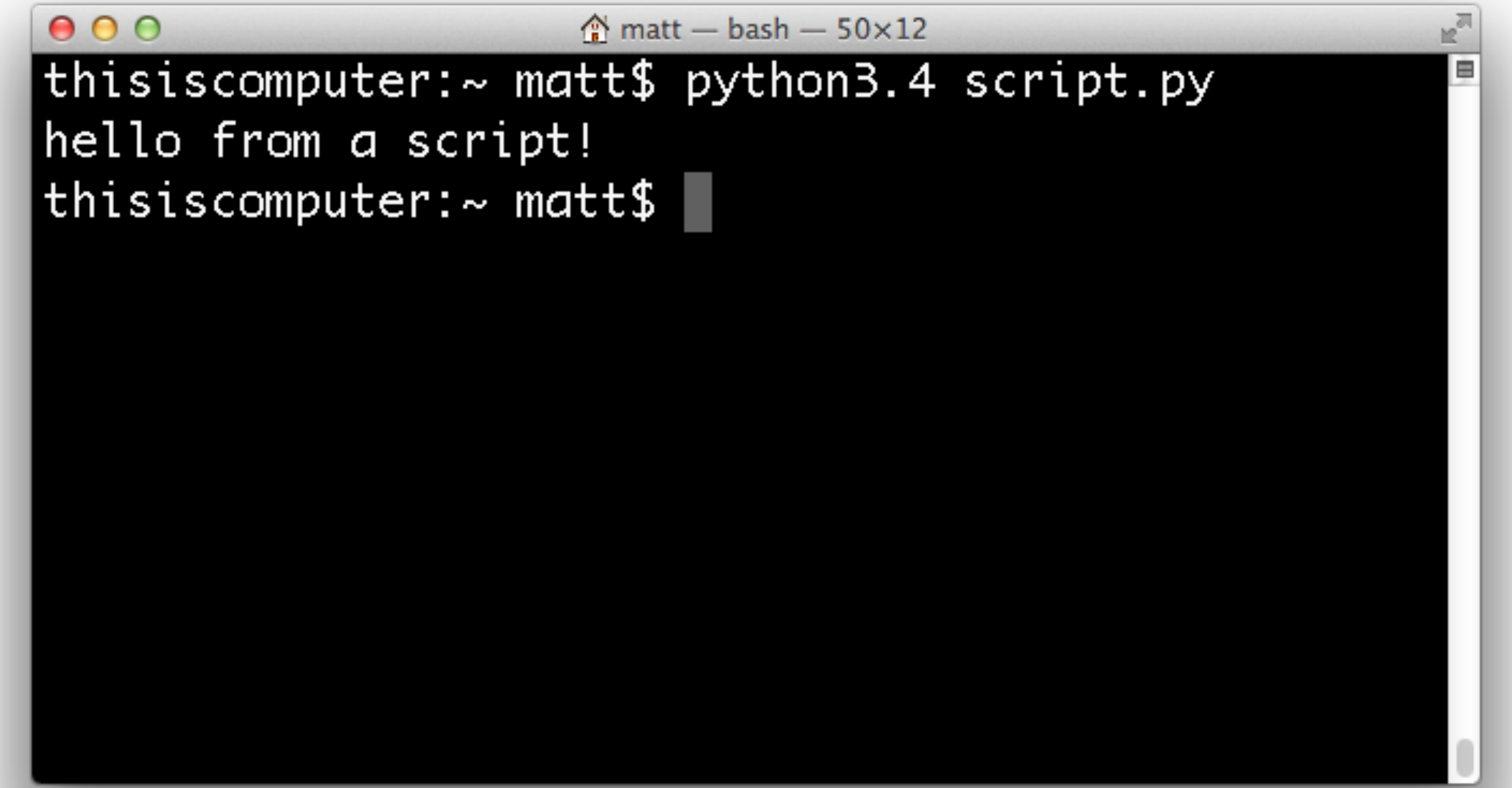
- Script Mode: specify the script to run.



```
thisiscomputer:~ matt$ python3.4 script.py
```

Python Interpreter

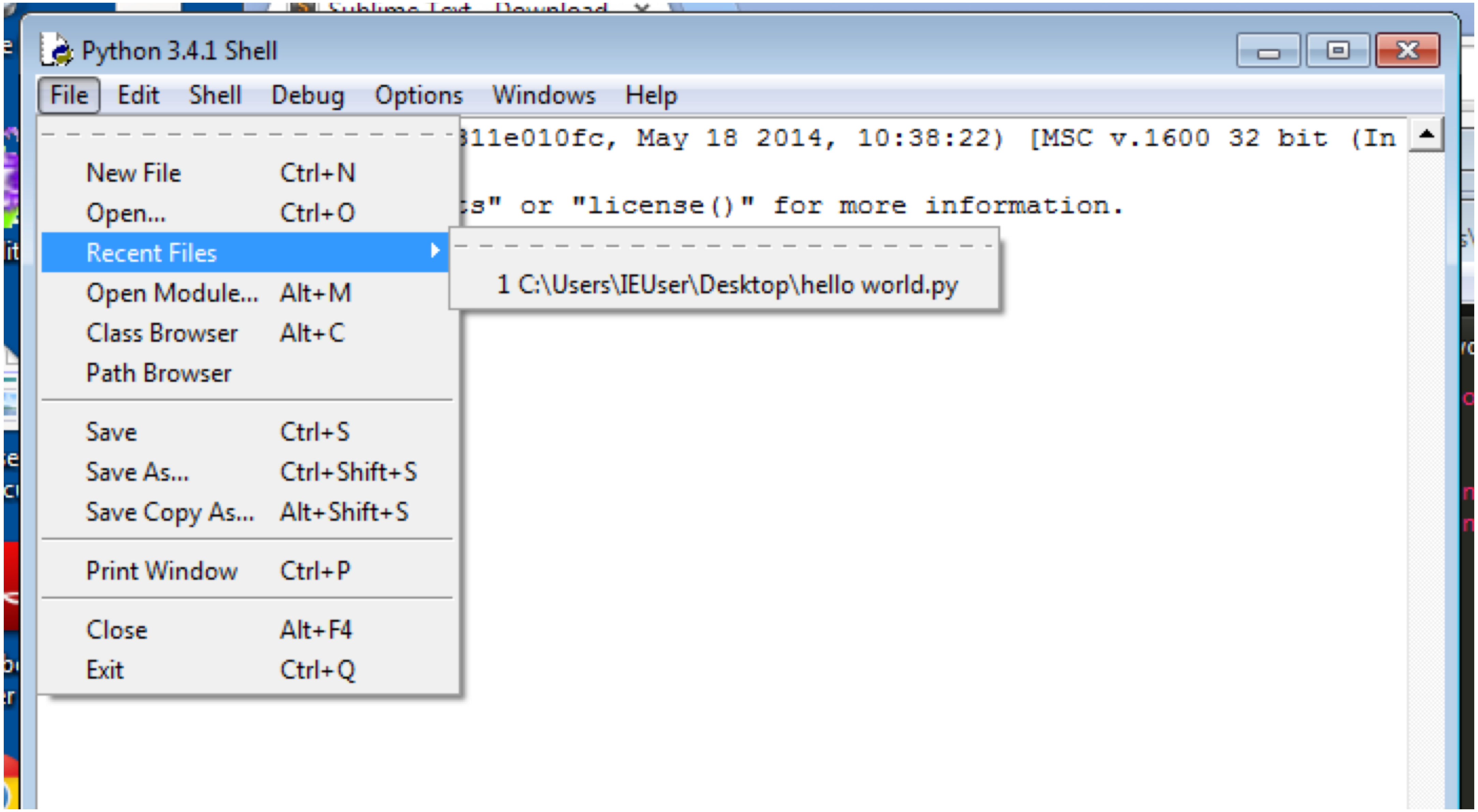
- Script Mode



```
thisiscomputer:~ matt$ python3.4 script.py
hello from a script!
thisiscomputer:~ matt$ 
```

Python Interpreter

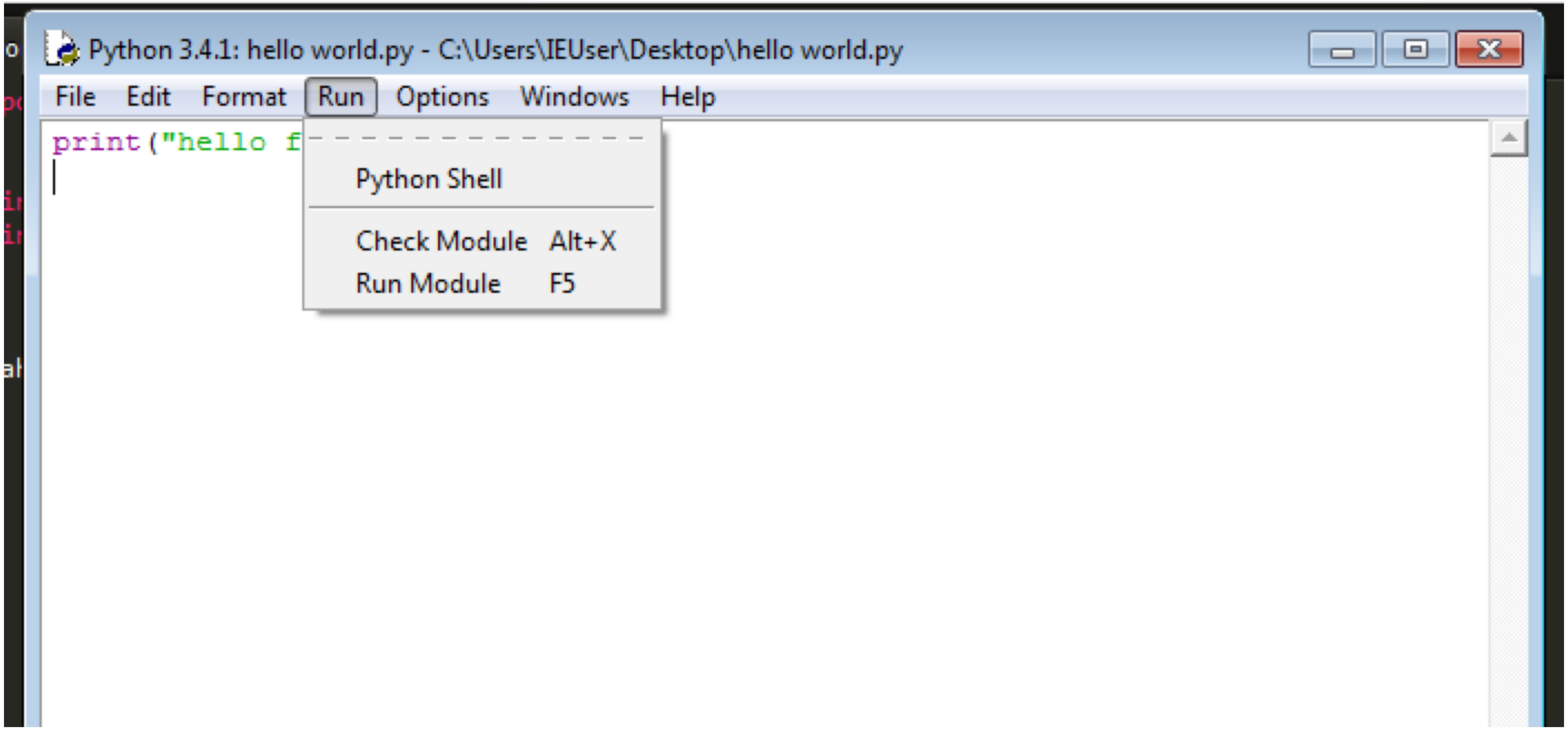
- Script Mode (Windows)



```
311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (In

s" or "license()" for more information.
```

Python Interpreter

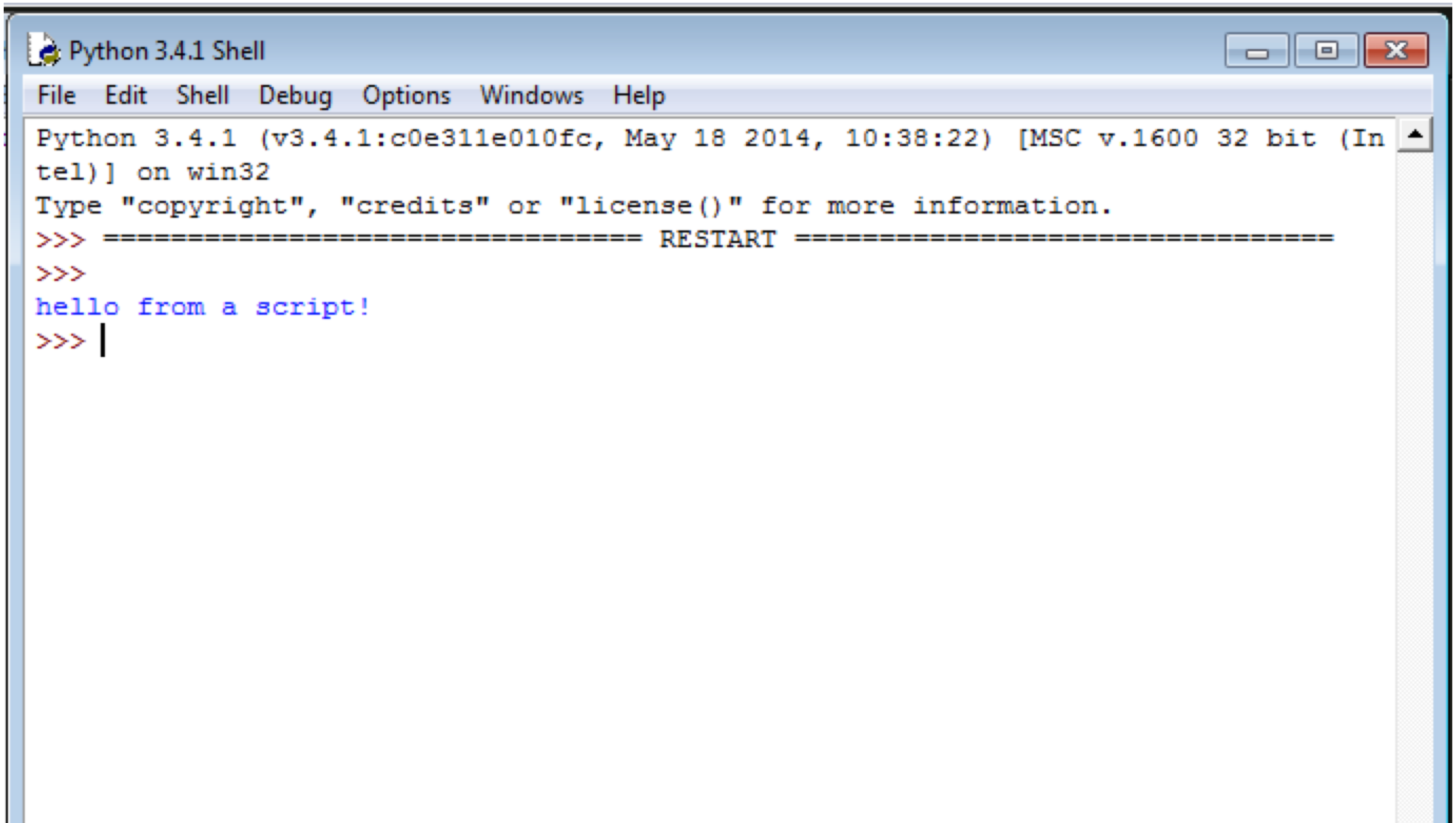
- Script Mode (Windows)



```
print("hello f
```

Python Interpreter

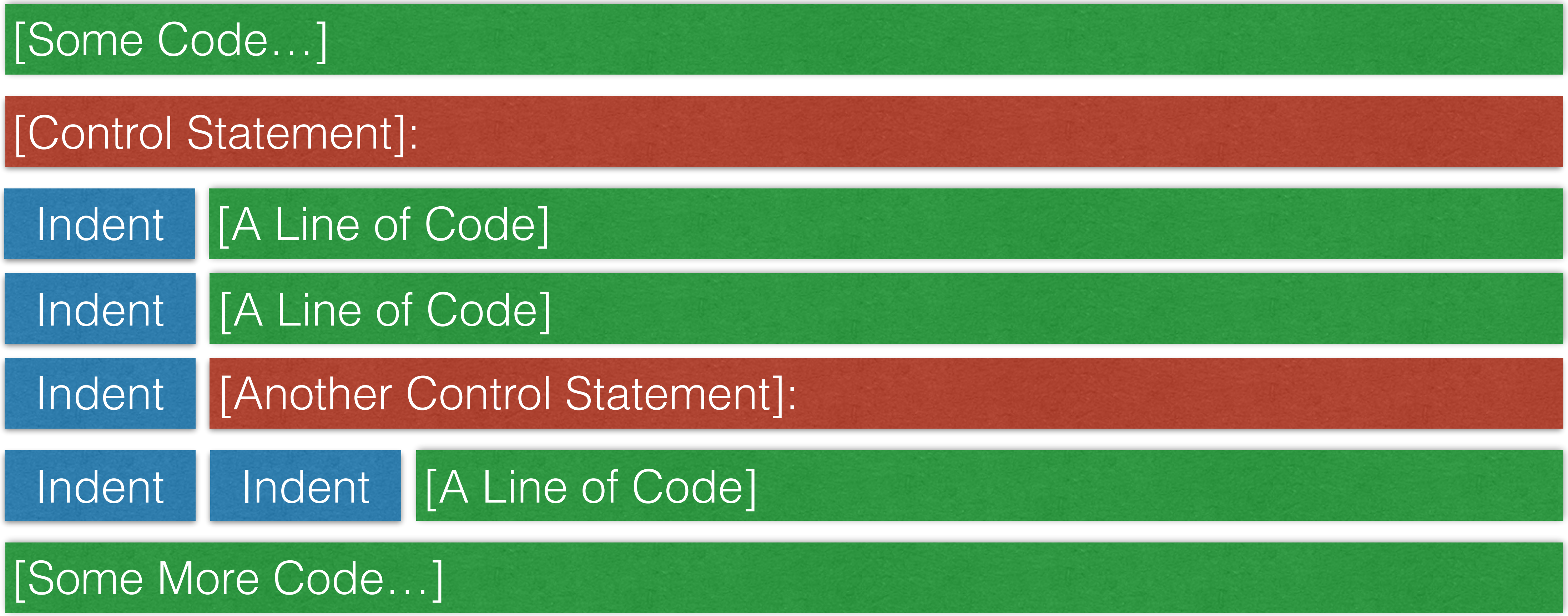
- Script Mode (Windows)



```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
hello from a script!
>>> 
```

Grammer

- Python relies on indentation and colons to control the flow of the script. Whitespace is important.



[Some Code…]

[Control Statement]:

Indent [A Line of Code]

Indent [A Line of Code]

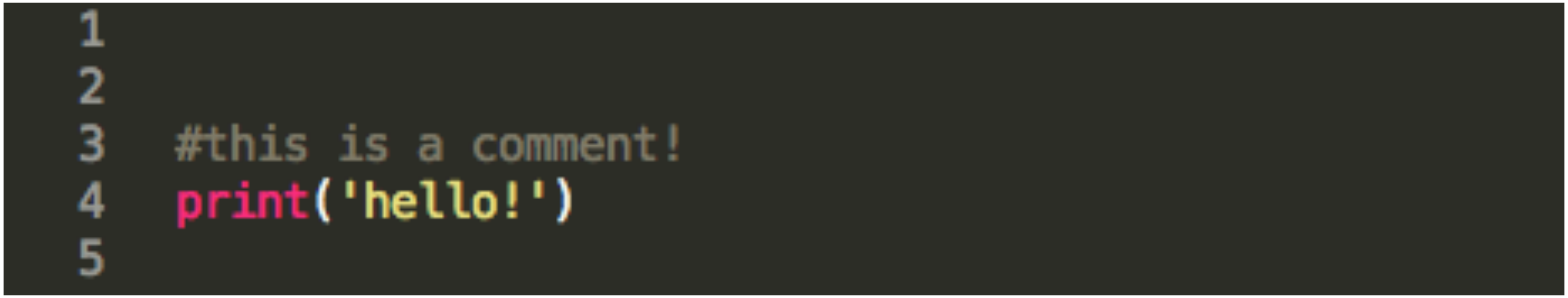
Indent [Another Control Statement]:

Indent Indent [A Line of Code]

[Some More Code…]

Grammer

- Add comments using #



```python


#this is a comment!
print('hello!')

```

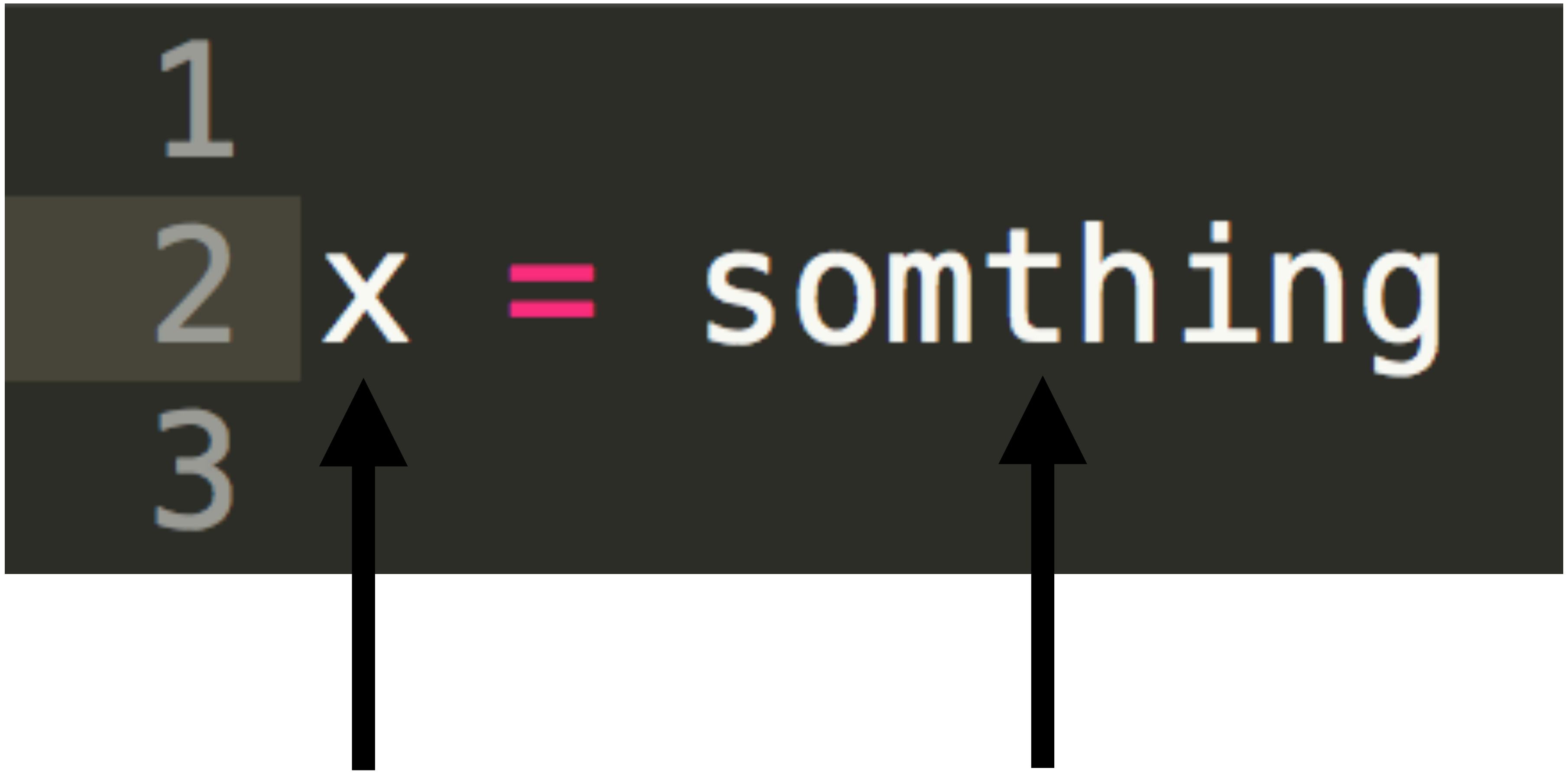
Grammer

- Conventions
  - Sticking with a specific style. For example using underscores in your variable names, etc.

Python Basics

- Variables
- Control Structures
- Methods & Functions

Variables

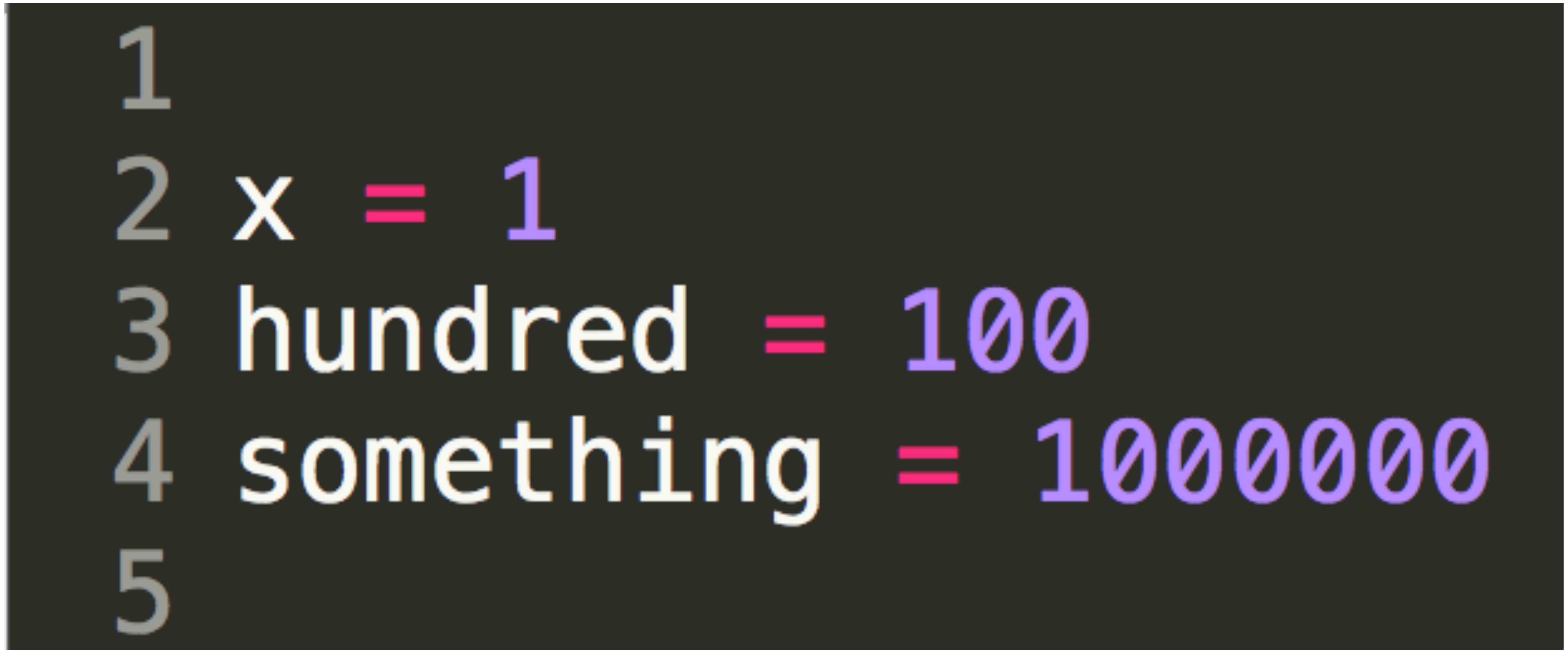


```
x = somthing
```

Your Label

The Value

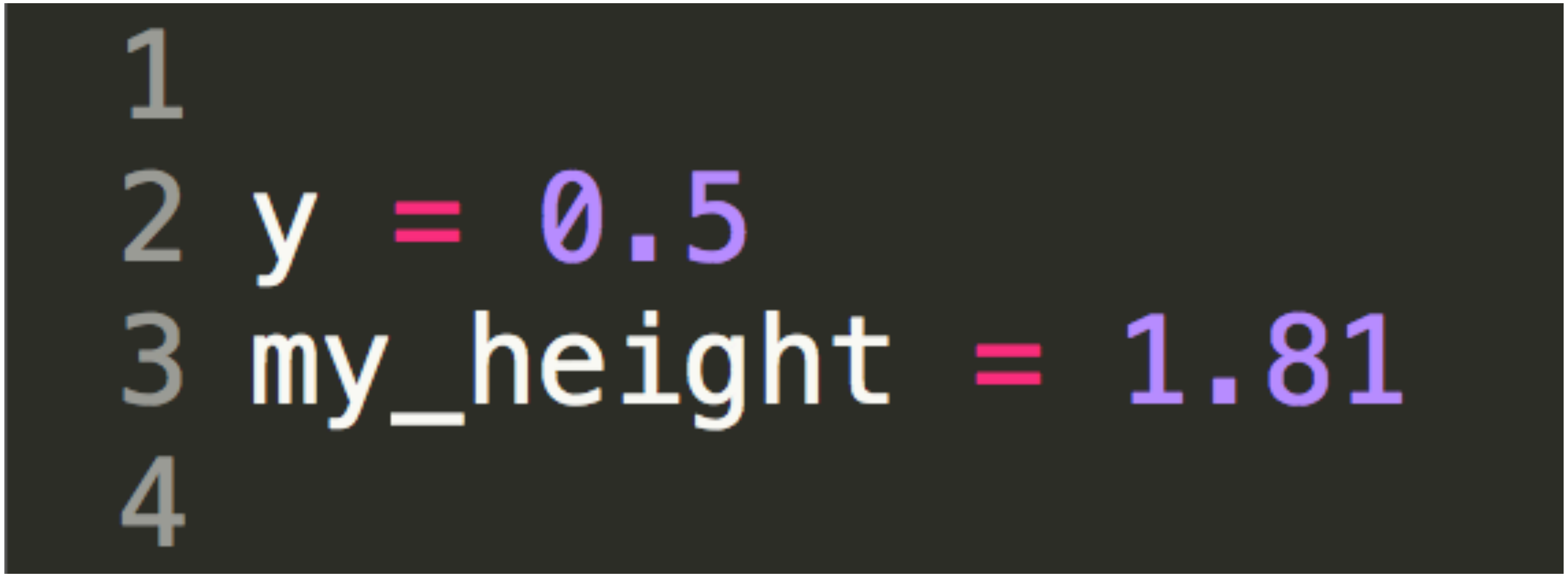
Variables - Integers



```
x = 1
hundred = 100
something = 1000000
```

Whole Numbers

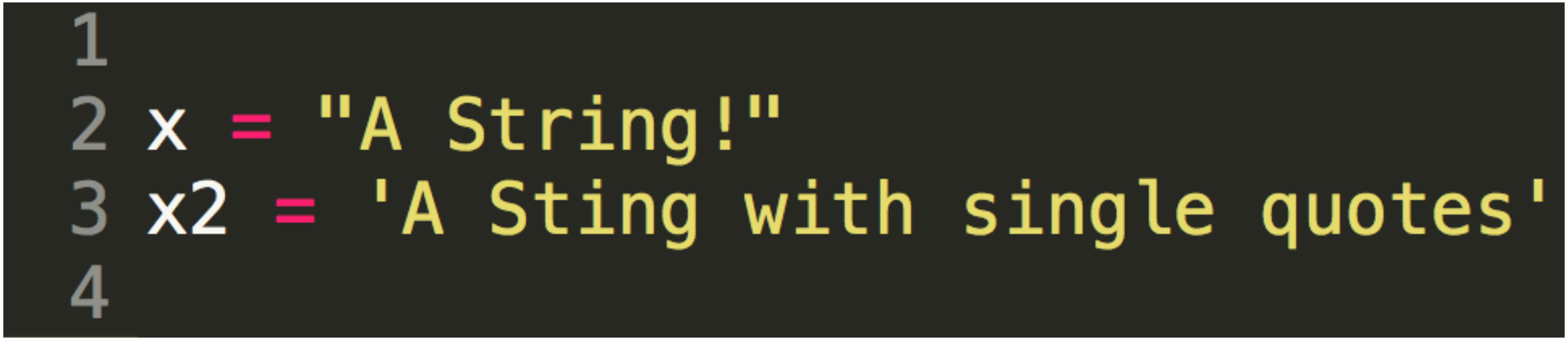
Variables - Floats



```

y = 0.5
my_height = 1.81

```

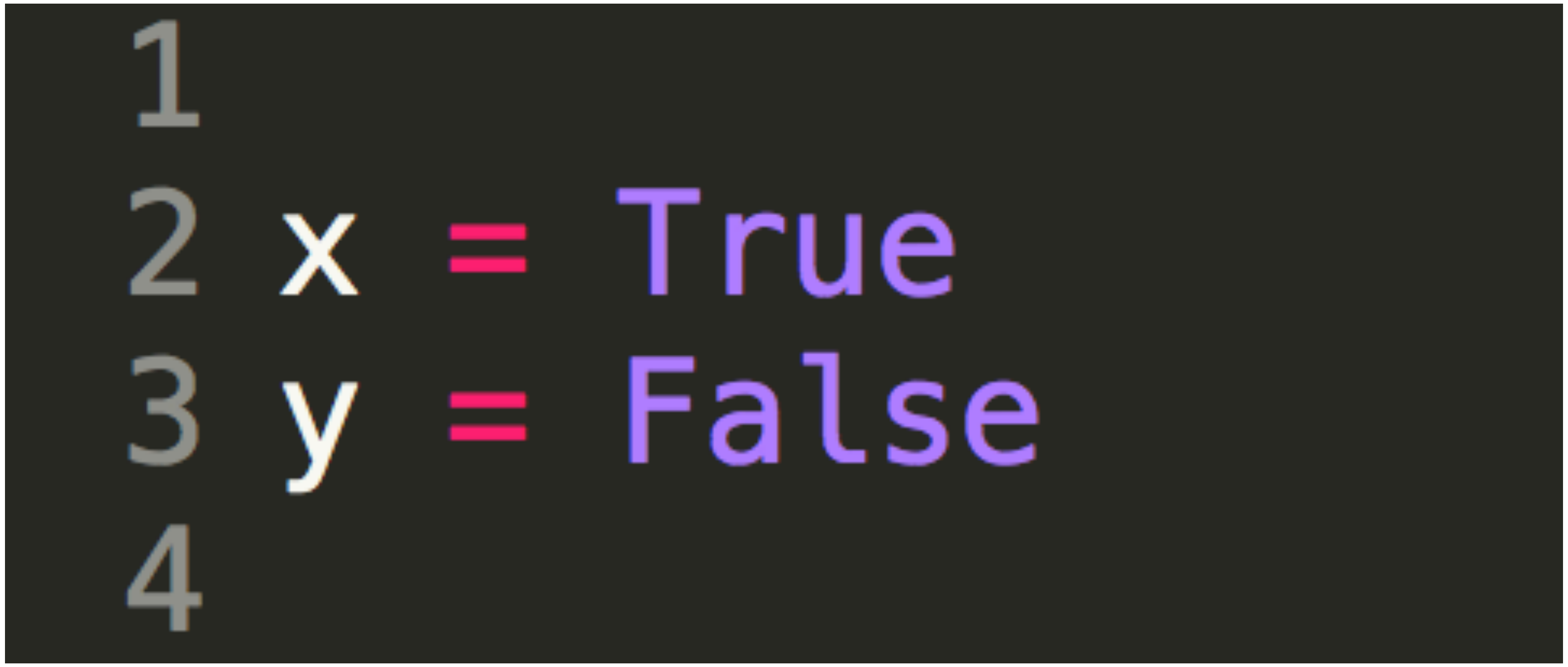
Decimal Numbers

Variables - Strings



```

x = "A String!"
x2 = 'A Sting with single quotes'

```

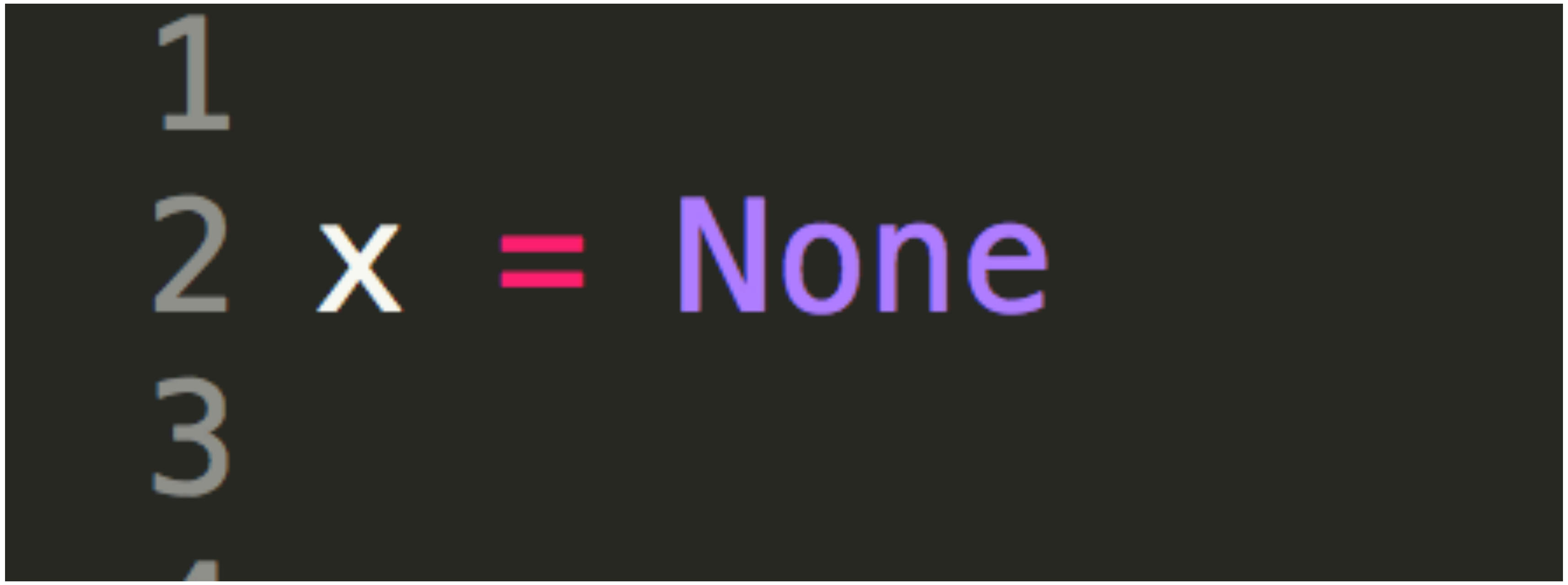
Variables - Boolean



```python
x = True
y = False
```

Only two possible values

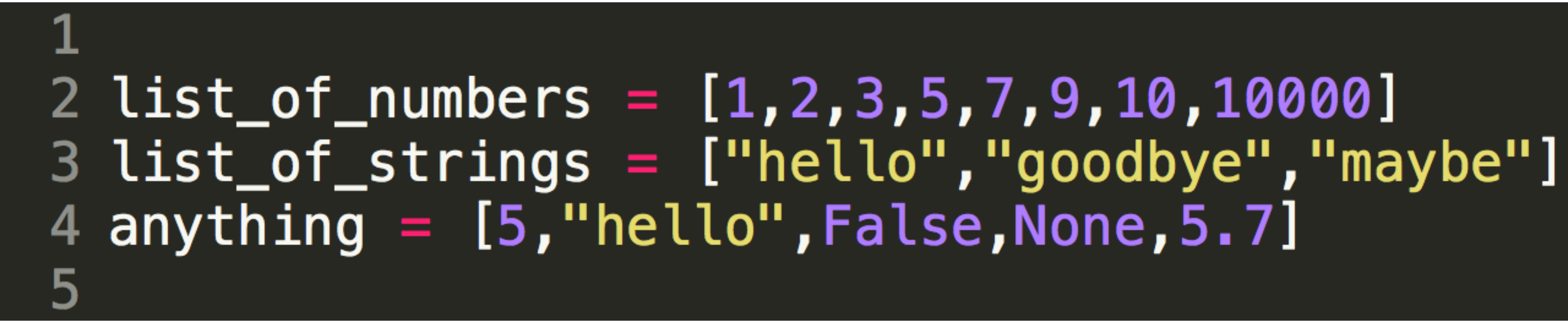
Variables - None



```python
x = None
```

Also known as: Null

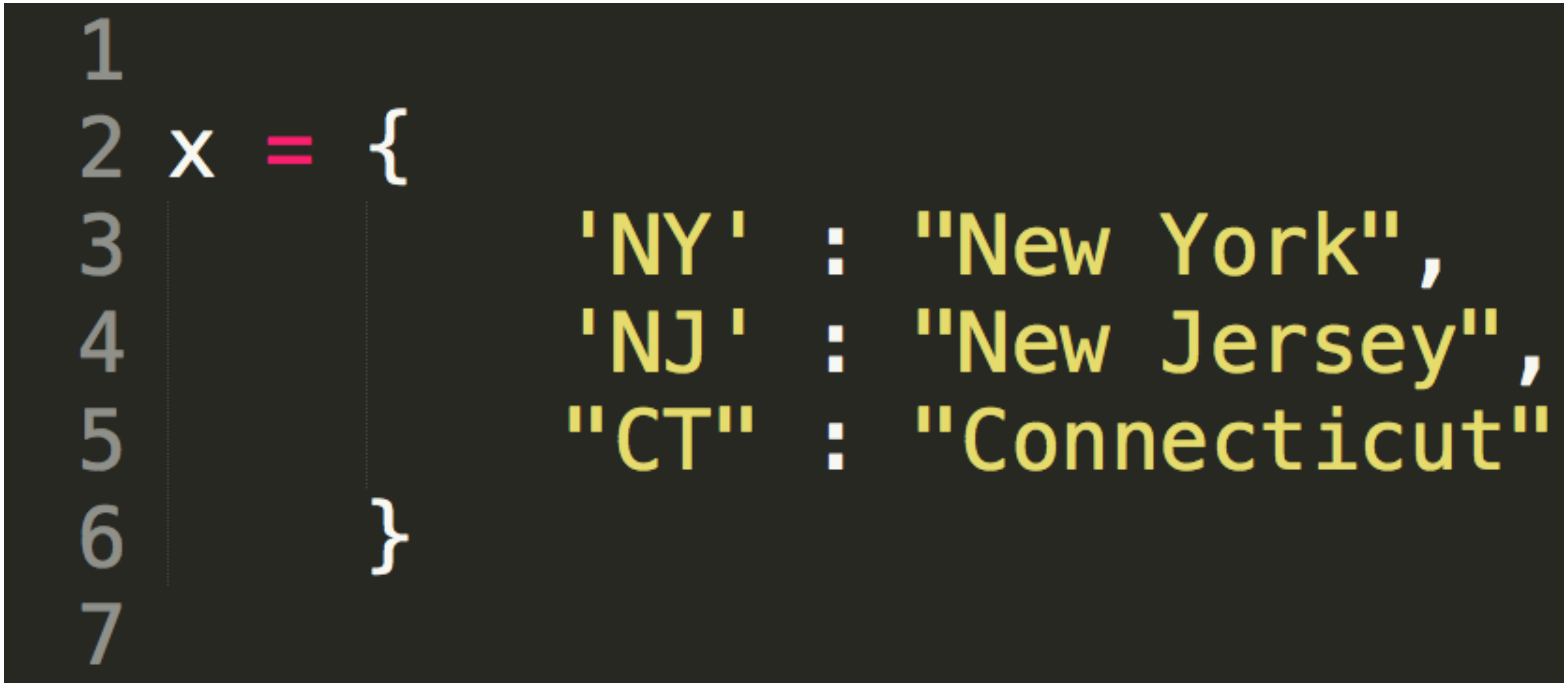
Complex Variables - List



```

list_of_numbers = [1,2,3,5,7,9,10,10000]
list_of_strings = ["hello","goodbye","maybe"]
anything = [5,"hello",False,None,5.7]

```

Also know as: Array
Look for the brackets: [ ]
Reference the things in the list using it’s index (x[0],x[5],etc)

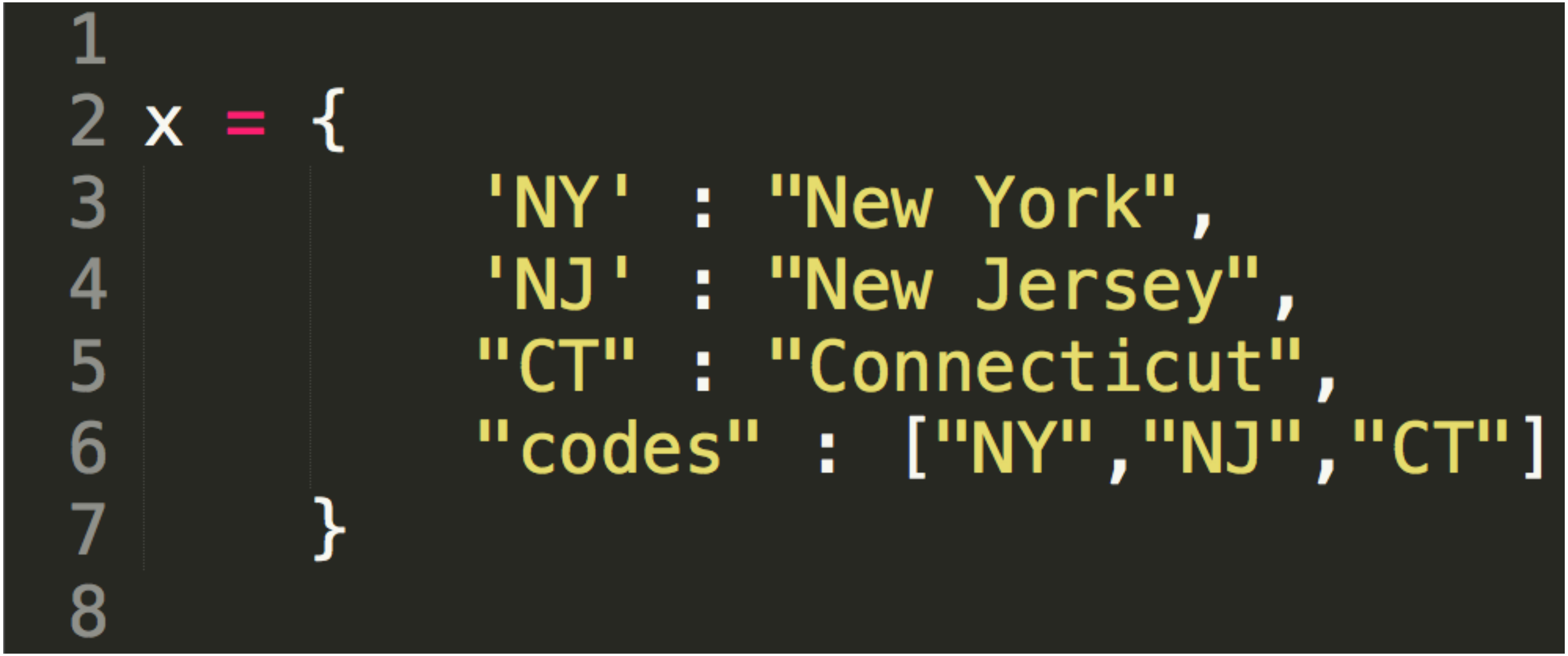
Complex Variables - Dictionary



```
x = {
        'NY' : "New York",
        'NJ' : "New Jersey",
        "CT" : "Connecticut"
    }
```

Also know as: Object
A Key/Value Store
Look for the curly brackets: {}

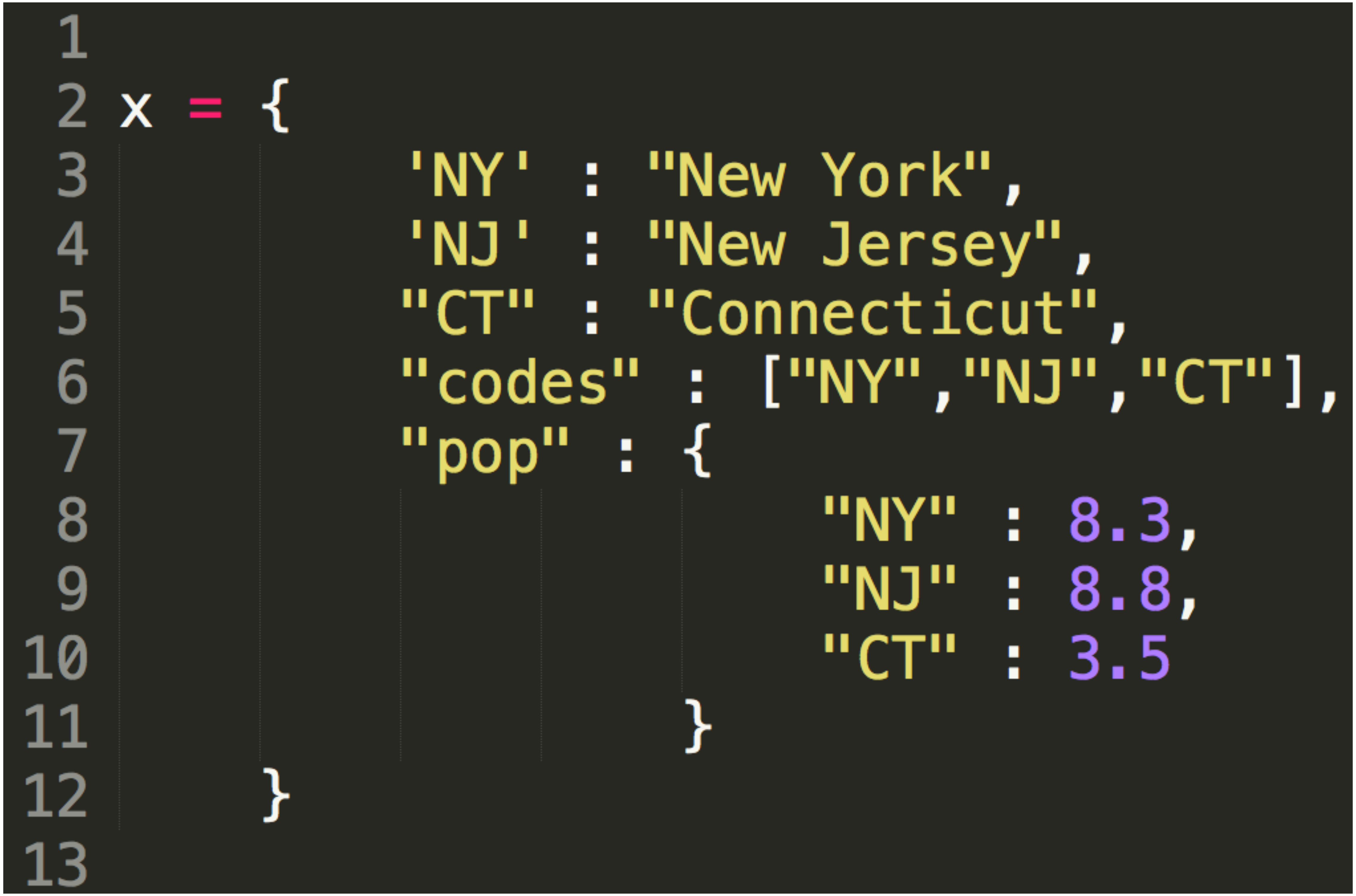
Complex Variables - Dictionary



```

x = {
        'NY' : "New York",
        'NJ' : "New Jersey",
        "CT" : "Connecticut",
        "codes" : ["NY","NJ","CT"]
    }

```

Also know as: Object
A Key/Value Store
Look for the curly brackets: {}

Complex Variables - Dictionary



```
x = {
    'NY' : "New York",
    'NJ' : "New Jersey",
    "CT" : "Connecticut",
    "codes" : ["NY","NJ","CT"],
    "pop" : {
          "NY" : 8.3,
          "NJ" : 8.8,
          "CT" : 3.5
        }
  }
```

Exercise - List

- Write a List (in your editor or on paper) of all the names of cats or dogs (or whatever) you know.

Exercise - List

```
animals = ["Bert","Miles","Rulla"]
```

Exercise - Dictionary

- Write a Dictionary (in your editor or on paper) where the name is the key and the value is the type of animal it is

Exercise - Dictionary

```
{
  "Bert" : "cat",
  "Miles" : "dog",
  "Rulla" : "cat"
}
```

Exercise - Dictionary

- Write a Dictionary (in your editor or on paper) where the name is the key and the value another dictionary with the type of animal and their age

Exercise - Dictionary

```
{
   "Bert" : {
             "type":"cat",
             "age" : 10
          },
   "Miles" : {
             "type":"dog",
             "age" : 8
          },
   "Rulla" : {
             "type":"cat",
             "age" : 10
          }
}
```

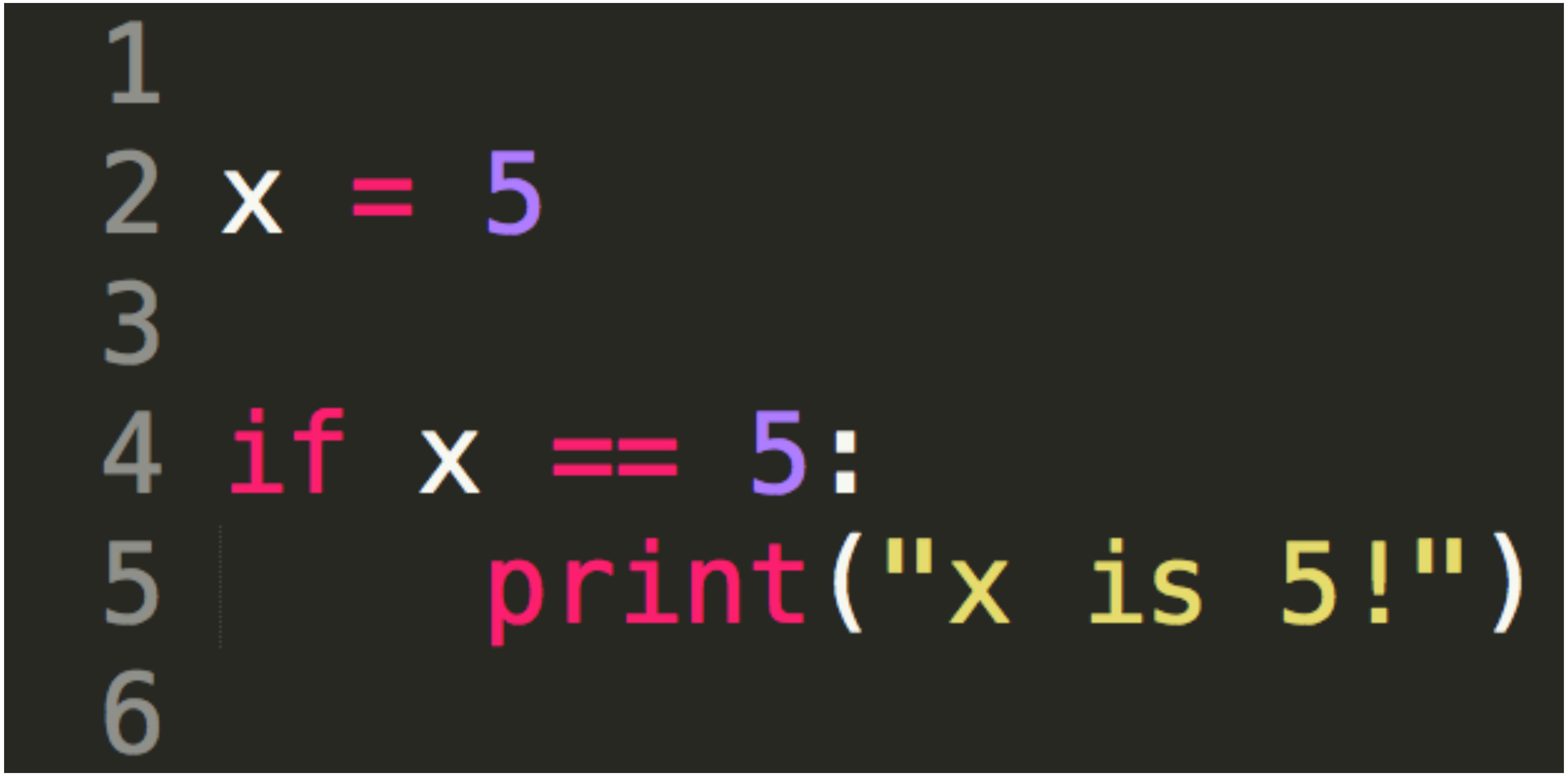
Exercise - Both

- Make a List of dictionaries that describes the animals name, type and age

Exercise - Dictionary

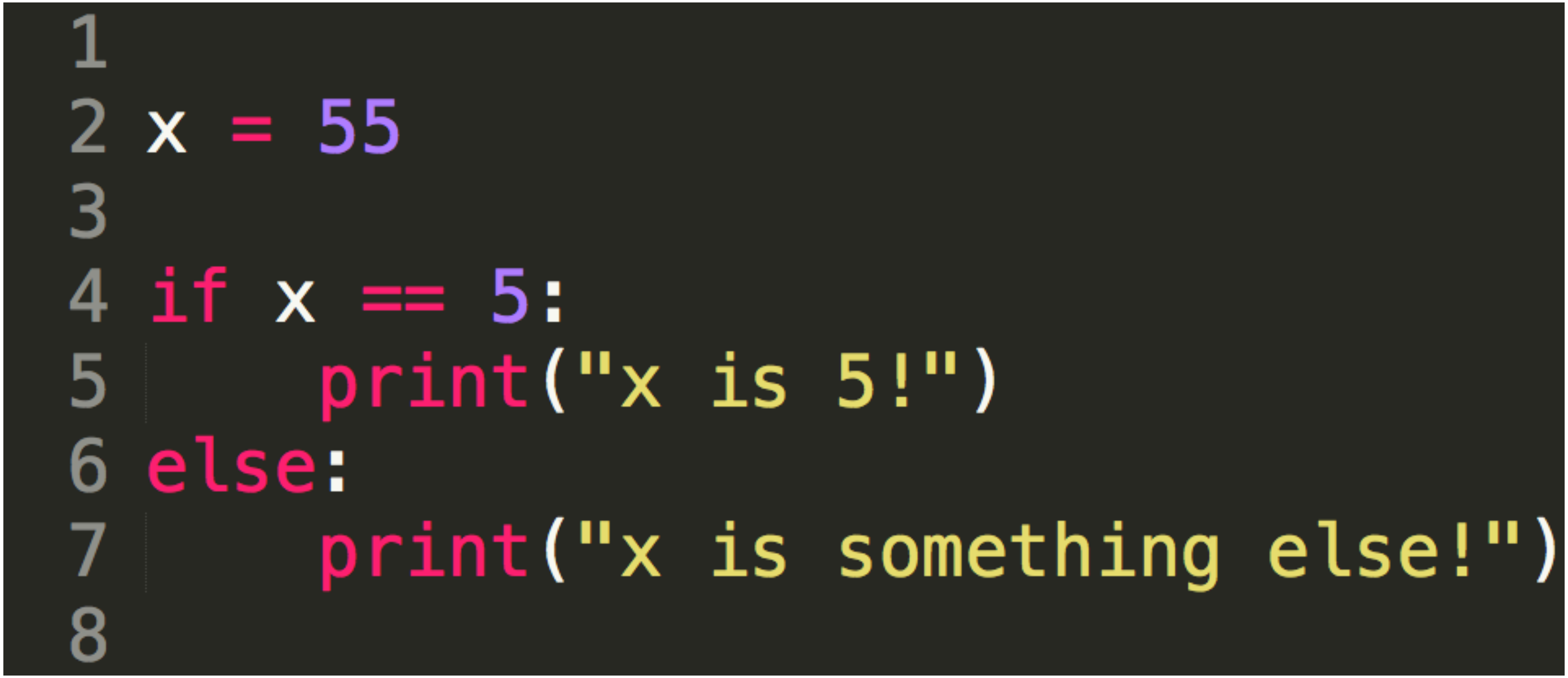
```
[
   {
      "name": "Bert",
      "type": "cat",
      "age" : 10
   },
   {
      "name": "Miles",
      "type": "dog",
      "age" : 8
   },
   {
      "name": "Rulla",
      "type": "cat",
      "age" : 10
   }
]
```

Control Structures - If Statement



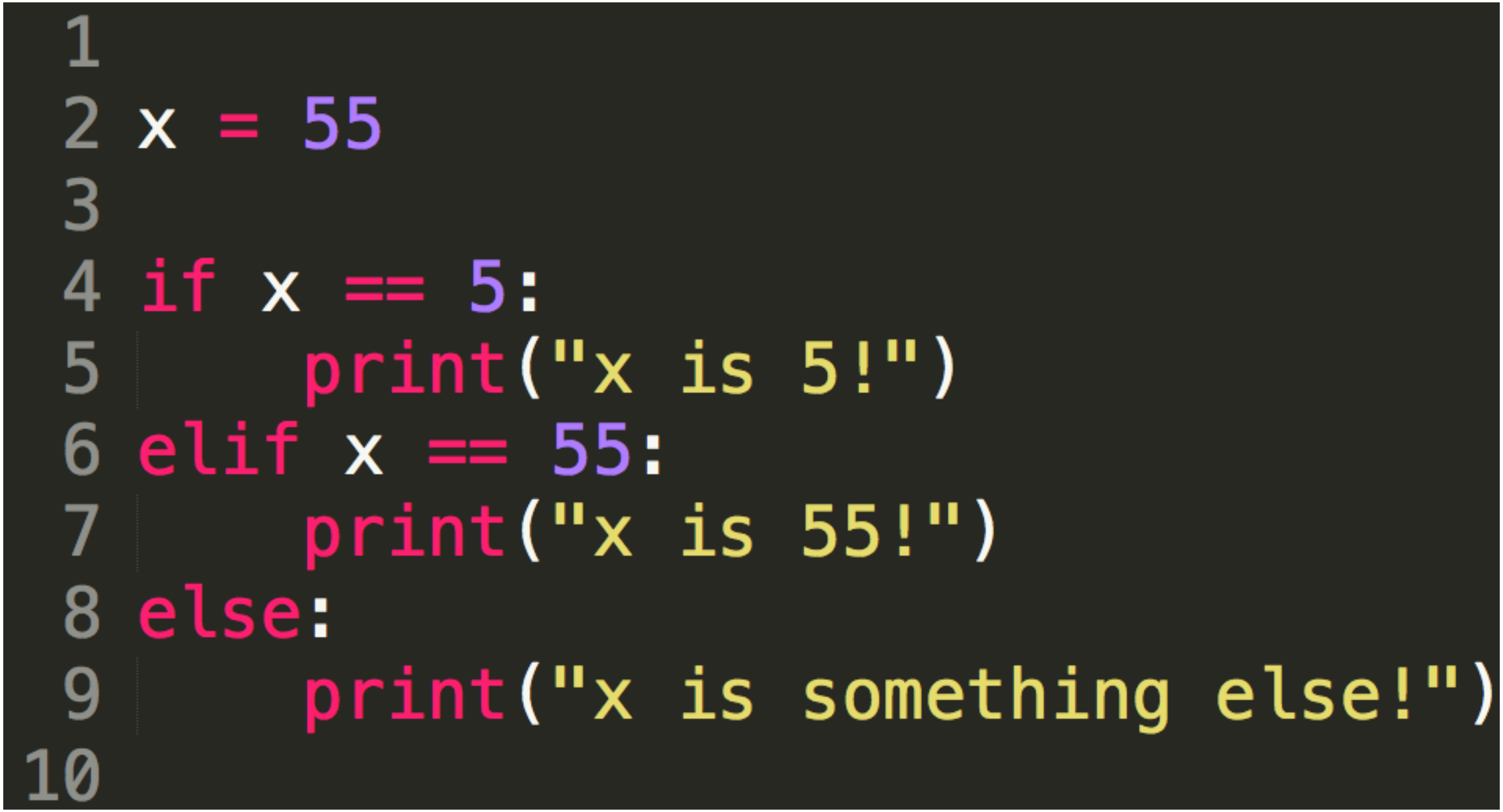
```python
x = 5

if x == 5:
    print("x is 5!")
```

Control Structures - If Else Statement



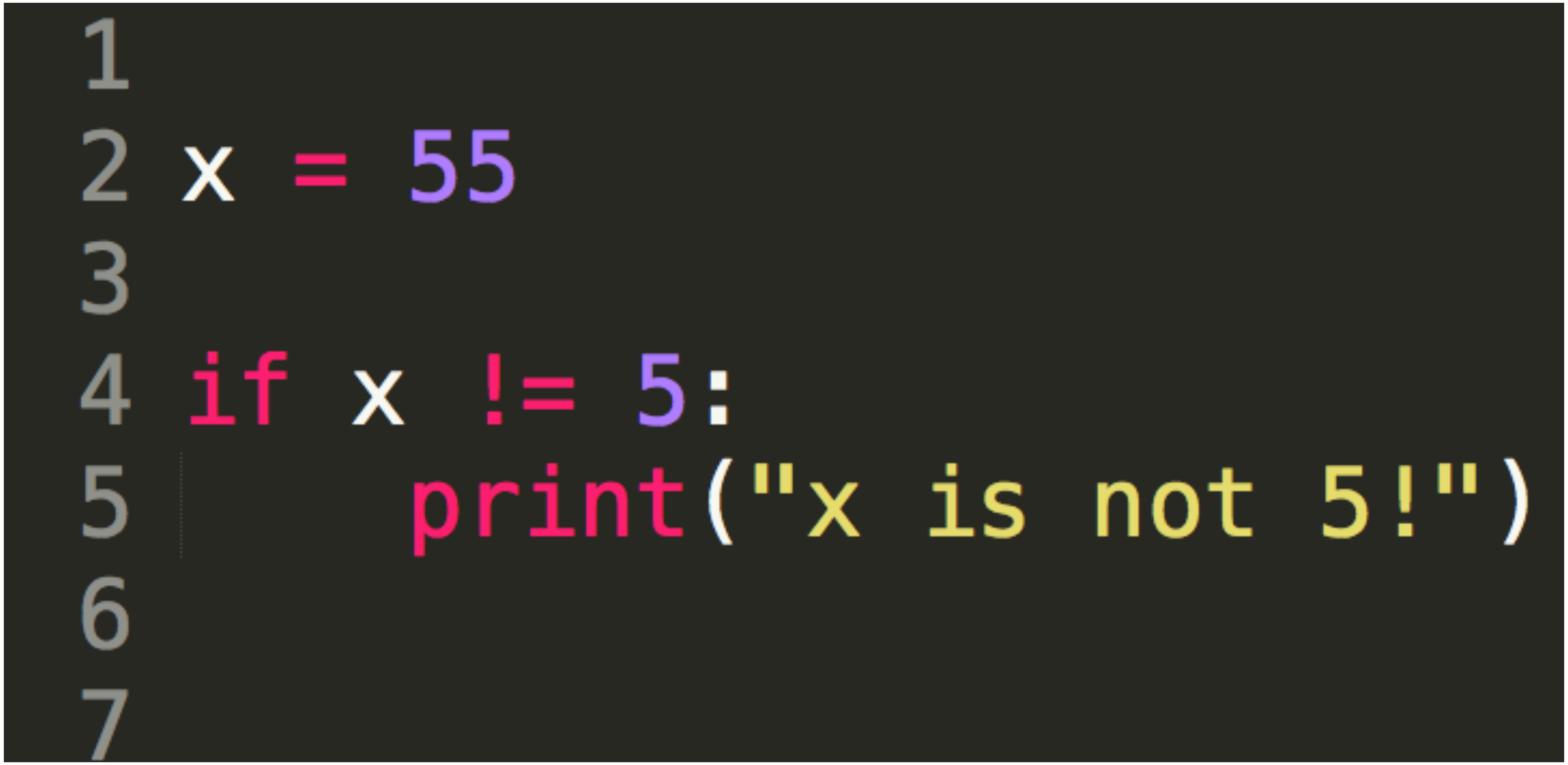
```python

x = 55

if x == 5:
	print("x is 5!")
else:
	print("x is something else!")

```

Control Structures - If Else Elif Statement



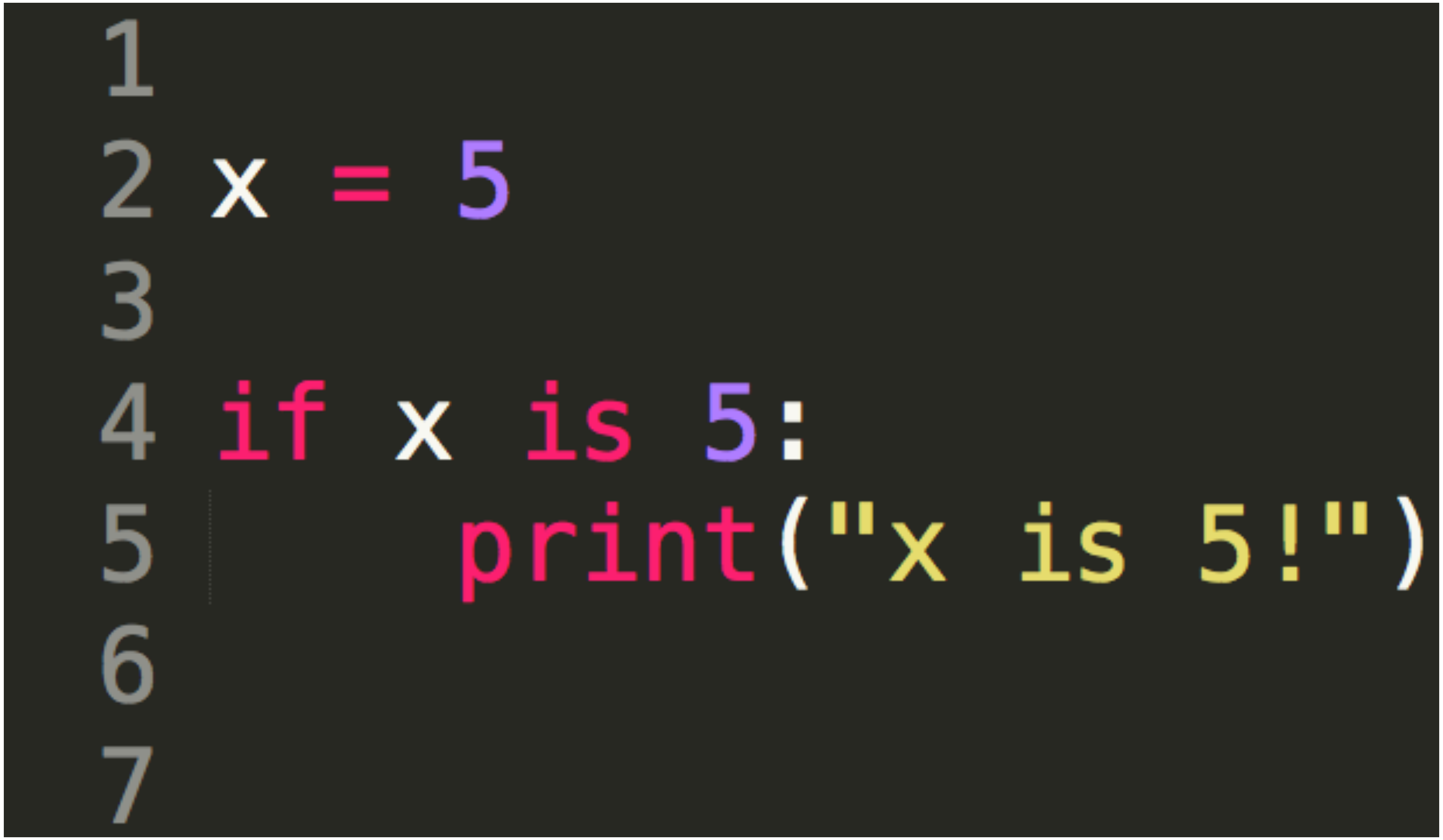
```python

x = 55

if x == 5:
    print("x is 5!")
elif x == 55:
    print("x is 55!")
else:
    print("x is something else!")

```

Control Structures -
If Statement



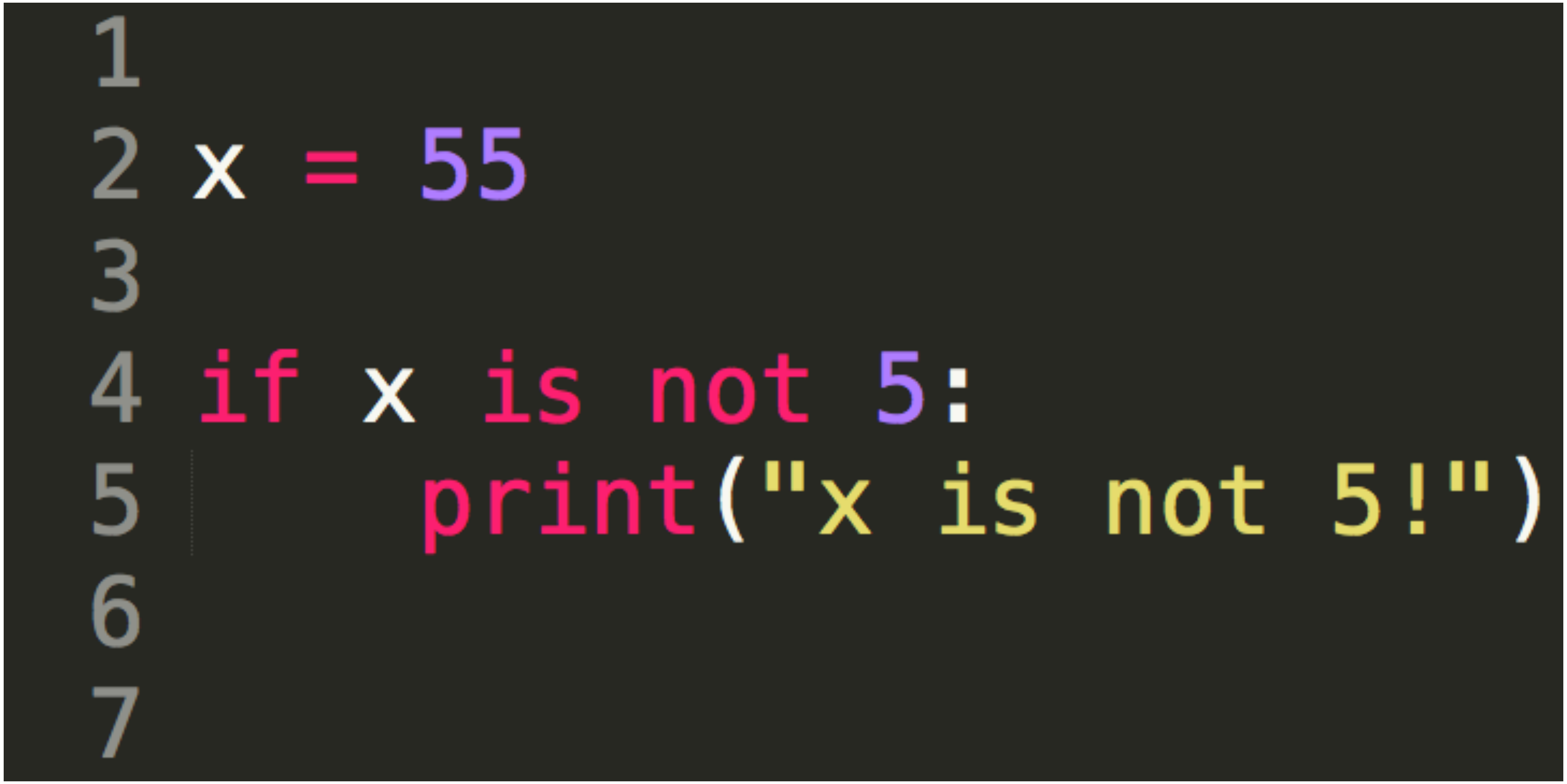
```python

x = 55

if x != 5:
    print("x is not 5!")


```

Control Structures - If Statement



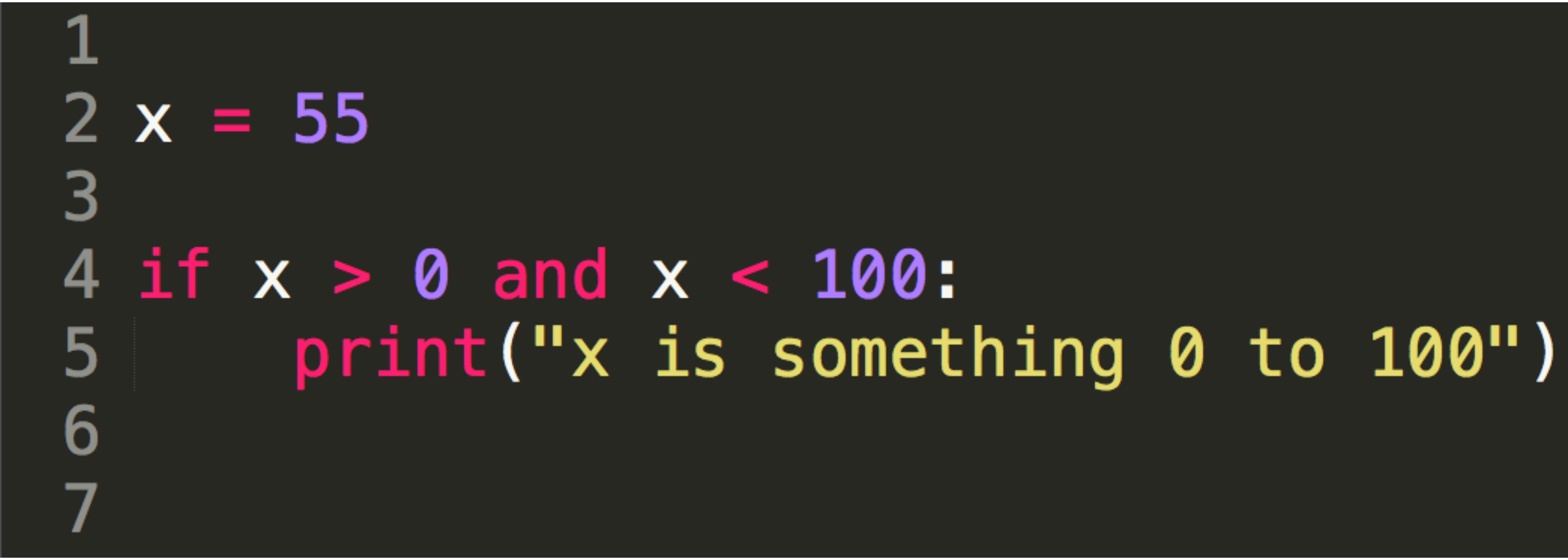
```

x = 5

if x is 5:
    print("x is 5!")


```

Control Structures - If Statement



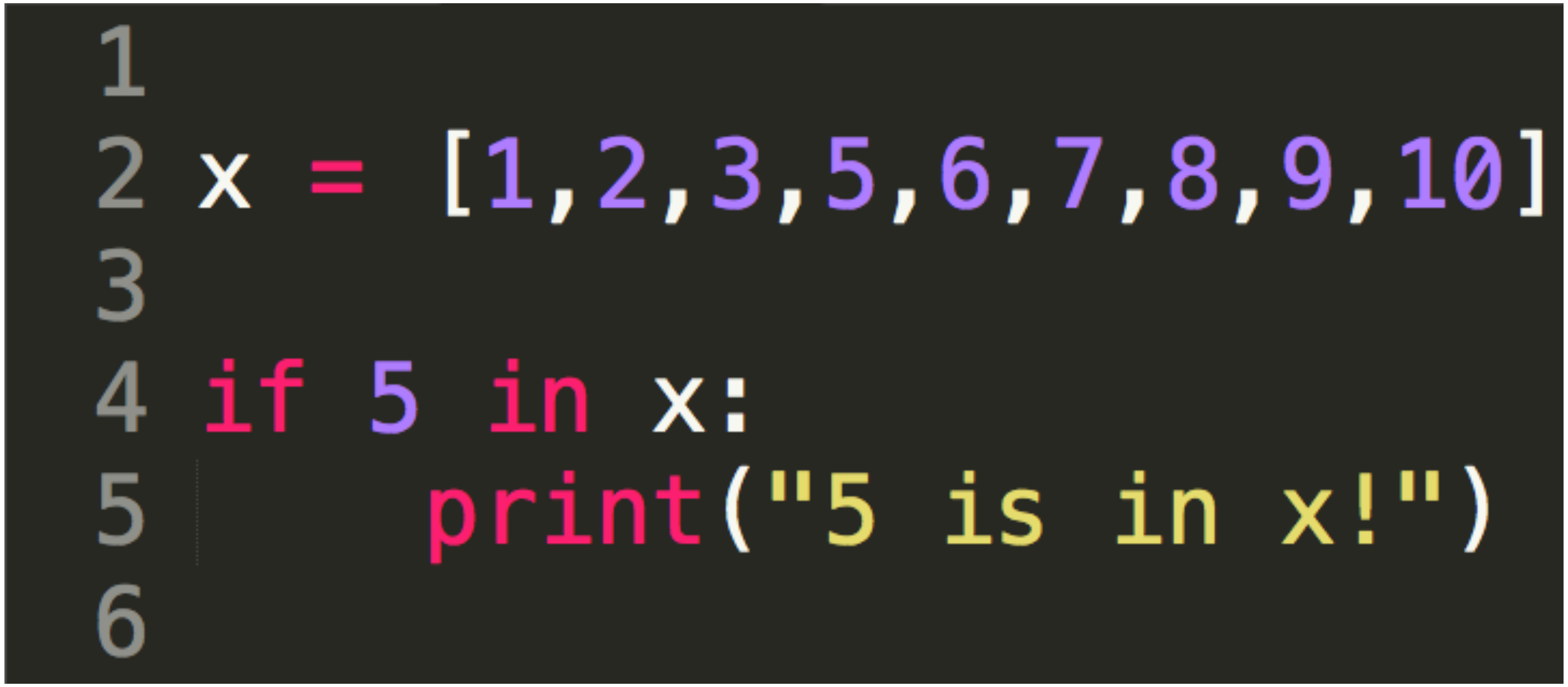
```python

x = 55

if x is not 5:
    print("x is not 5!")
```

Control Structures - If Statement

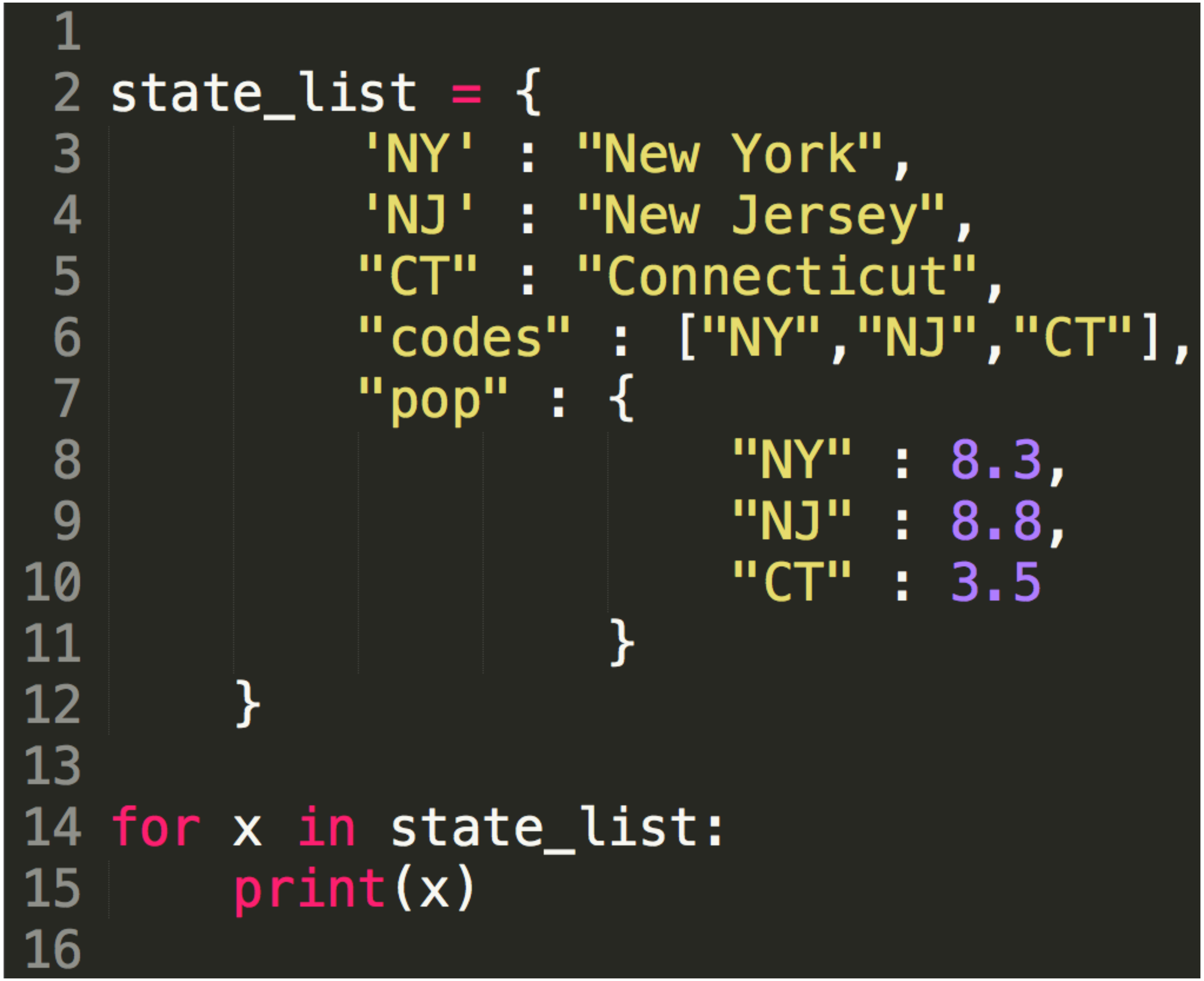


```

x = 55

if x > 0 and x < 100:
    print("x is something 0 to 100")


```

Control Structures - If Statement

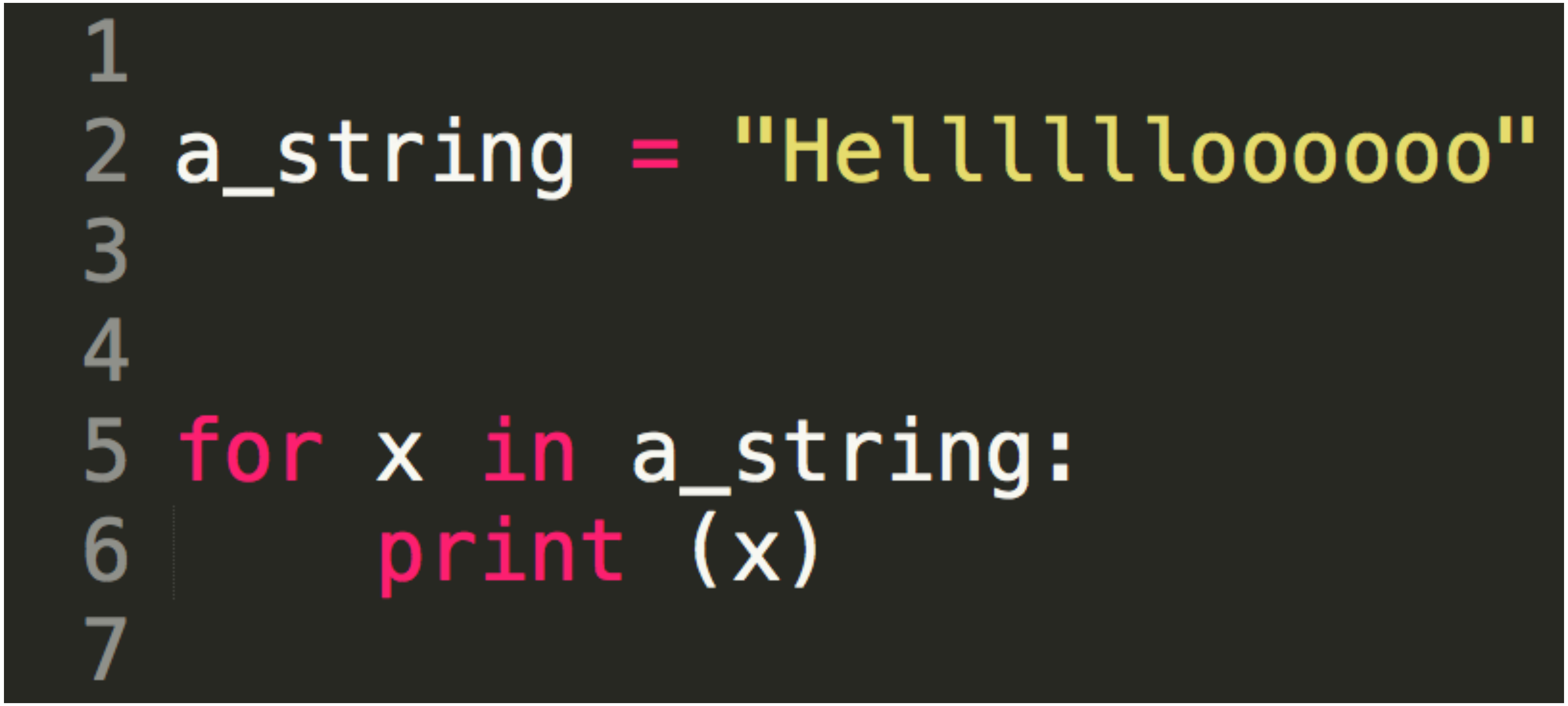


```

x = [1,2,3,5,6,7,8,9,10]

if 5 in x:
    print("5 is in x!")

```



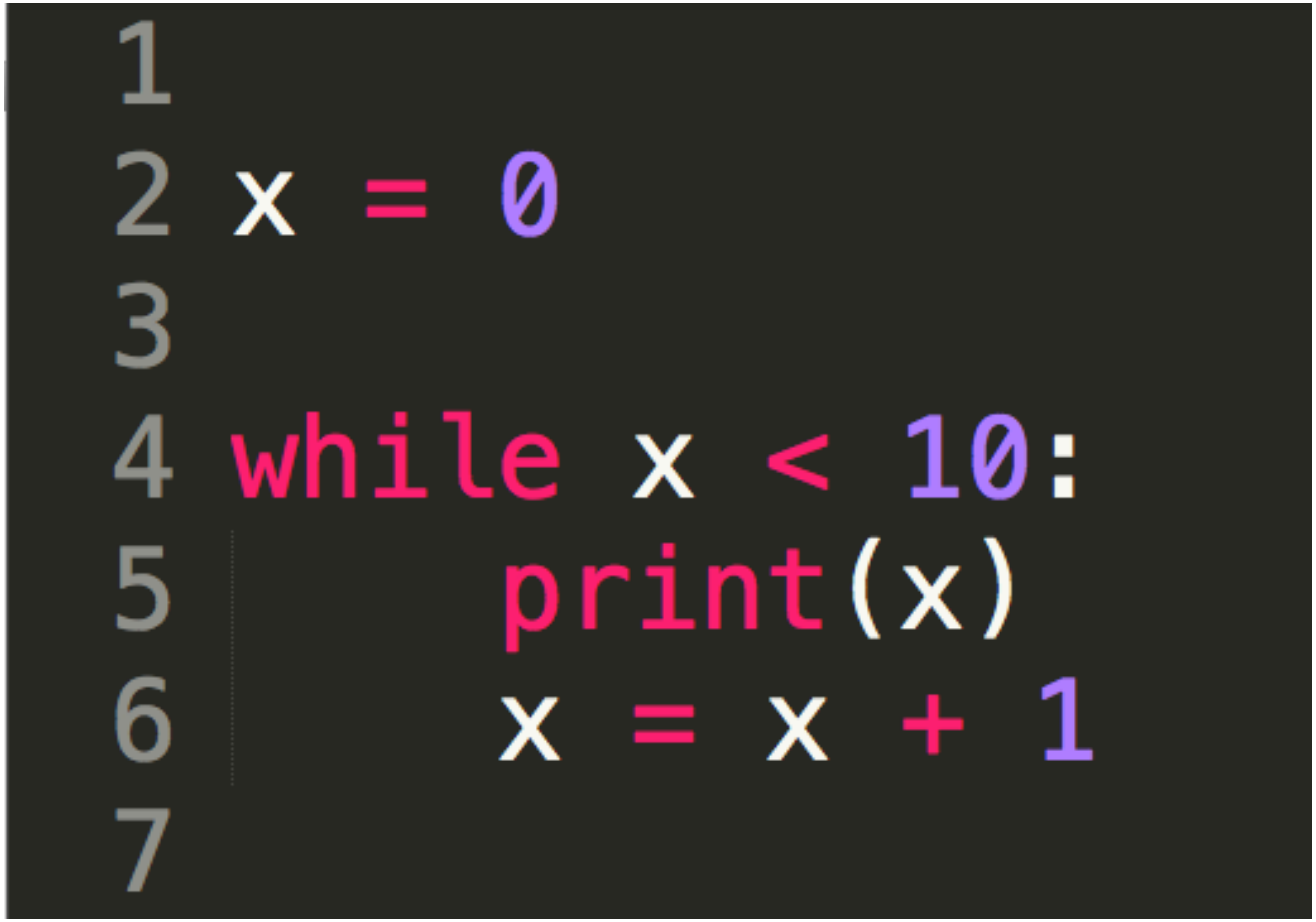
```python
state_list = {
        'NY' : "New York",
        'NJ' : "New Jersey",
        "CT" : "Connecticut",
        "codes" : ["NY","NJ","CT"],
        "pop" : {
                    "NY" : 8.3,
                    "NJ" : 8.8,
                    "CT" : 3.5
                }
    }

for x in state_list:
    print(x)
```

Control Structures - For Statement



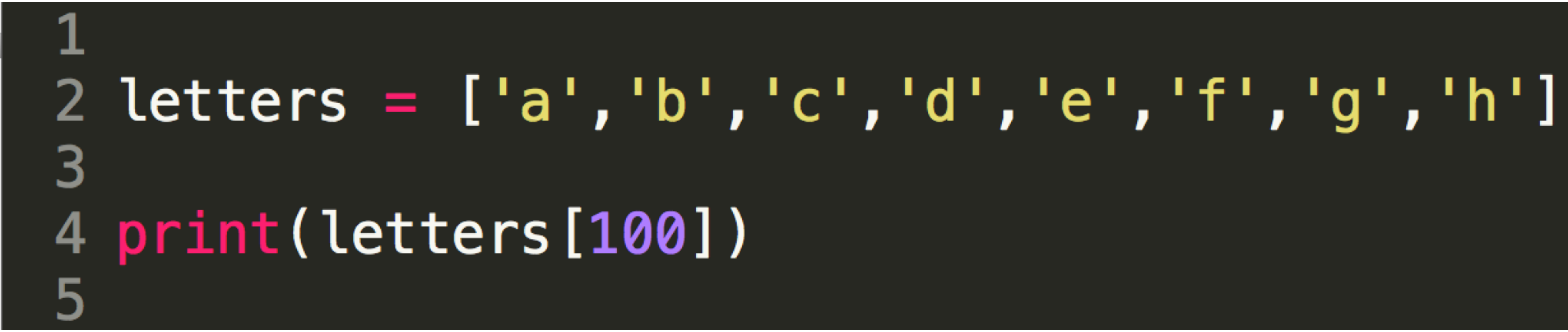
```python

a_string = "Helllllloooooo"


for x in a_string:
    print (x)

```

Control Structures - While Loop



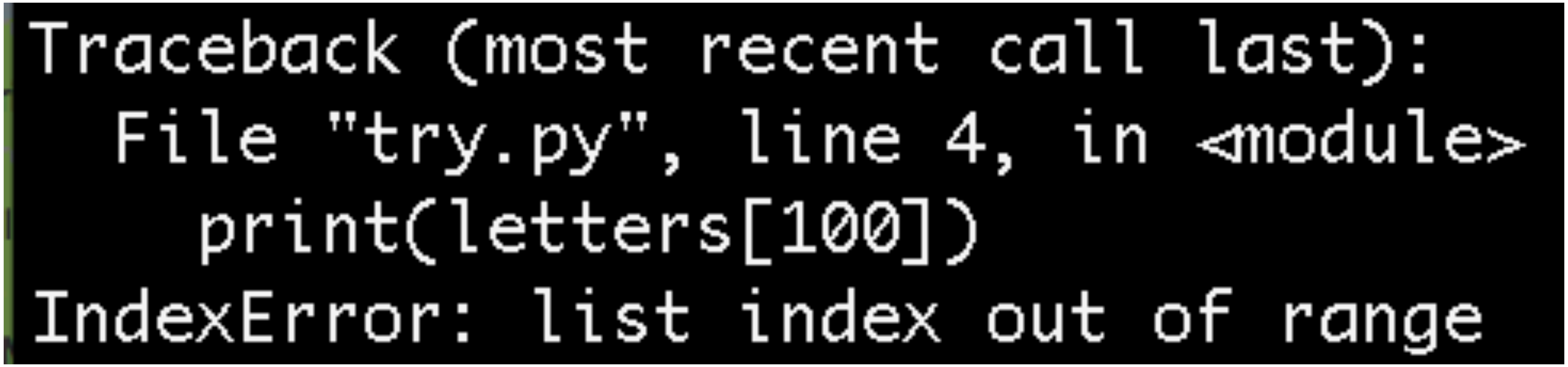
```
x = 0

while x < 10:
    print(x)
    x = x + 1
```

Control Structures - Try Statement



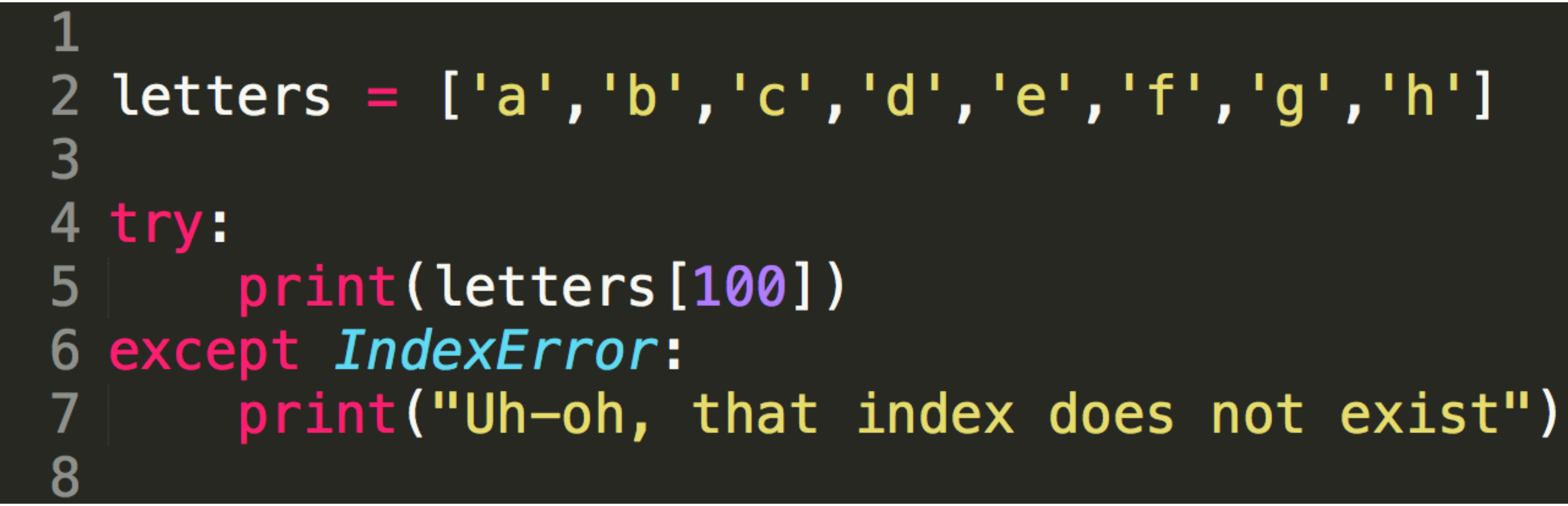
```python

letters = ['a','b','c','d','e','f','g','h']

print(letters[100])

```

Control Structures - Try Statement



```
Traceback (most recent call last):
  File "try.py", line 4, in <module>
    print(letters[100])
IndexError: list index out of range
```

Control Structures - Try Statement



```python

letters = ['a','b','c','d','e','f','g','h']

try:
    print(letters[100])
except IndexError:
    print("Uh-oh, that index does not exist")

```

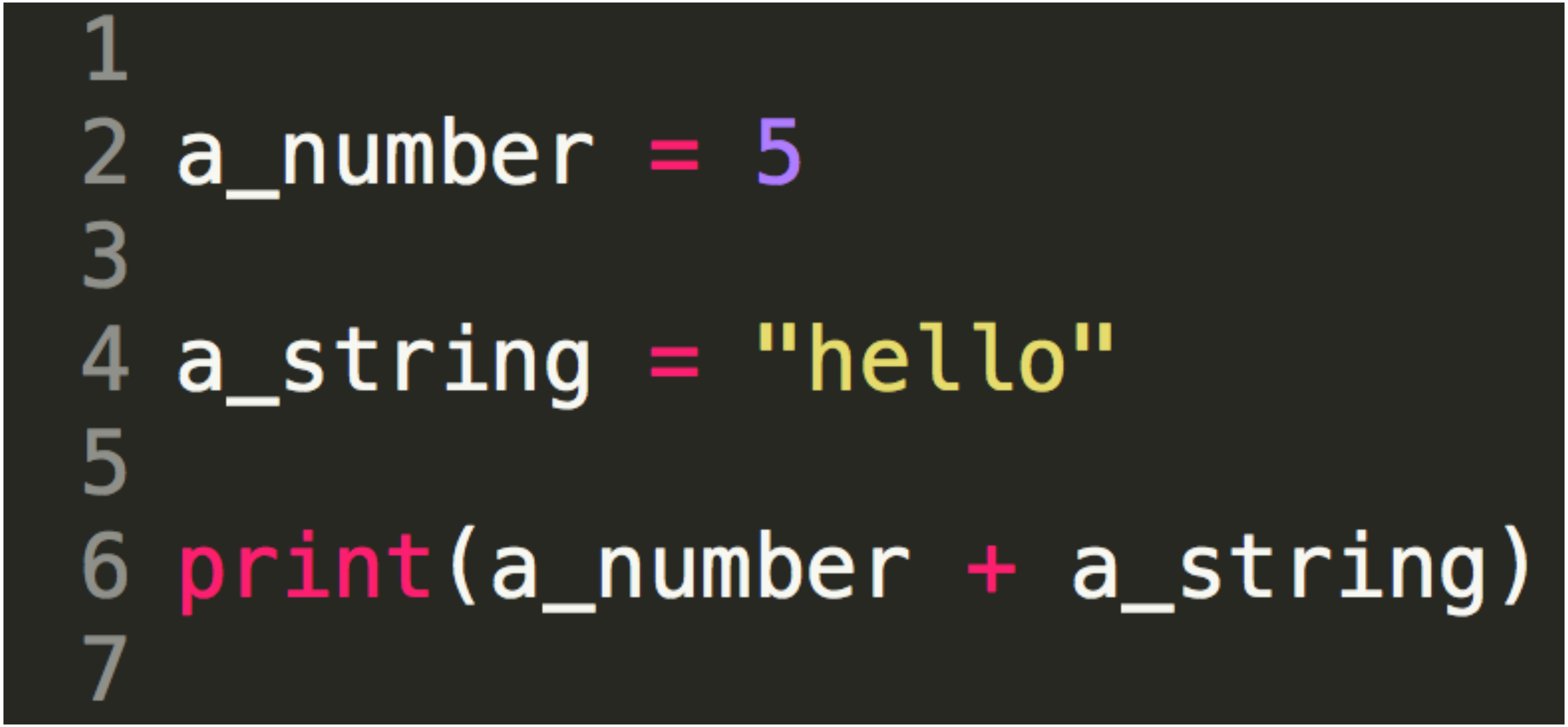
Functions

- Defined code that accepts parameters and returns a values. They can be “built-in”, written by you or others.

Functions

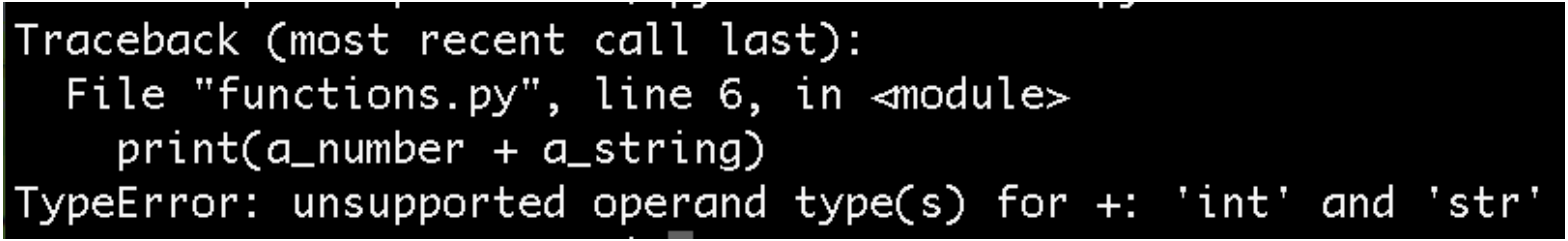
- Defined code that accepts parameters and returns a values. They can be “built-in”, written by you or others.

Functions



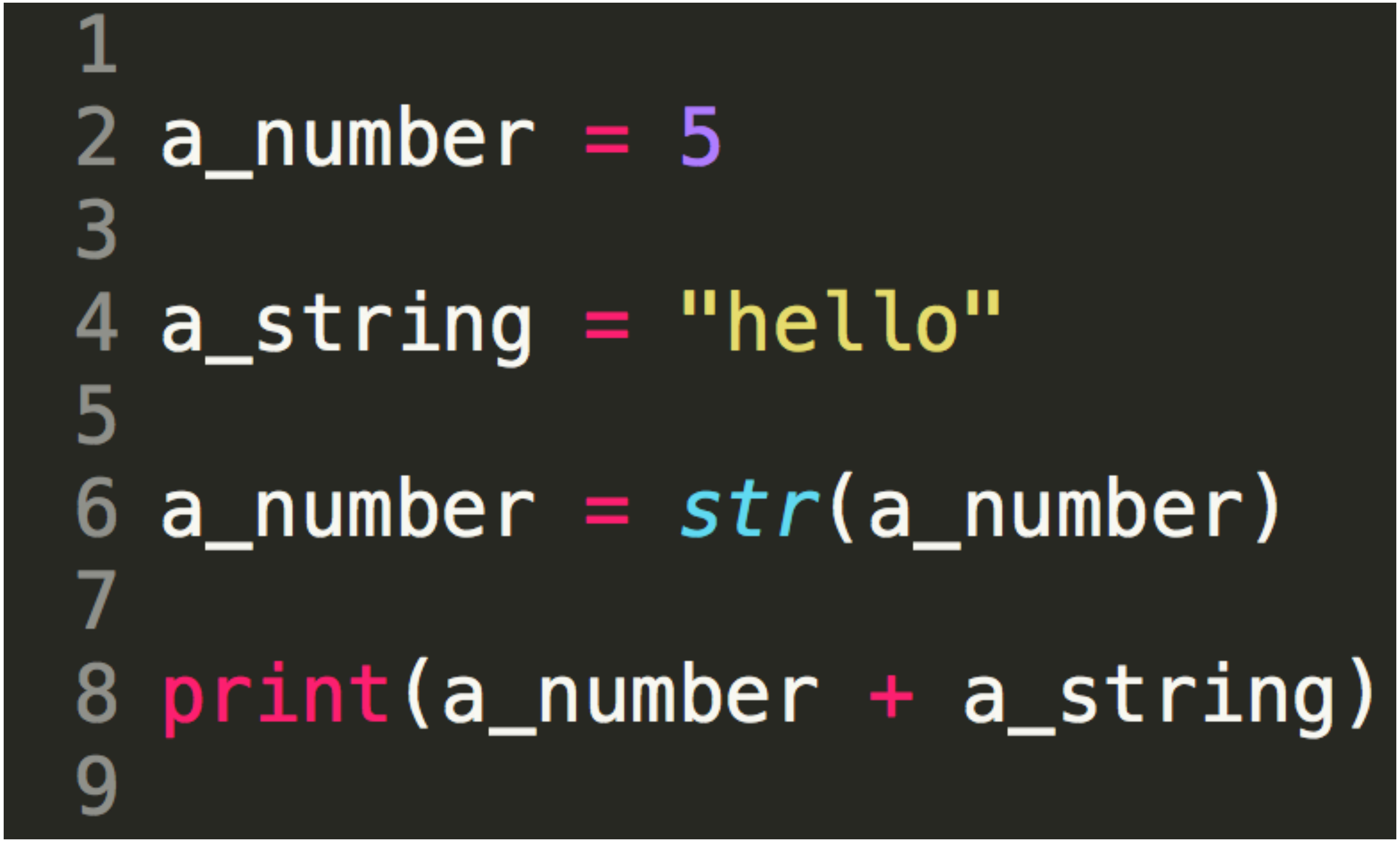
```

a_number = 5

a_string = "hello"

print(a_number + a_string)

```

Functions



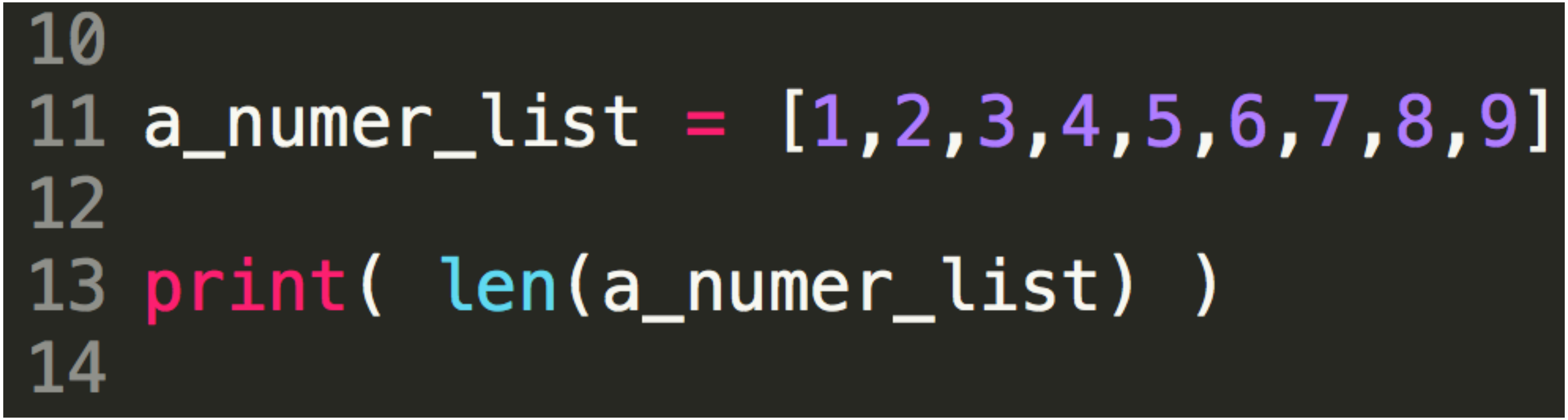
```
Traceback (most recent call last):
  File "functions.py", line 6, in <module>
    print(a_number + a_string)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Functions



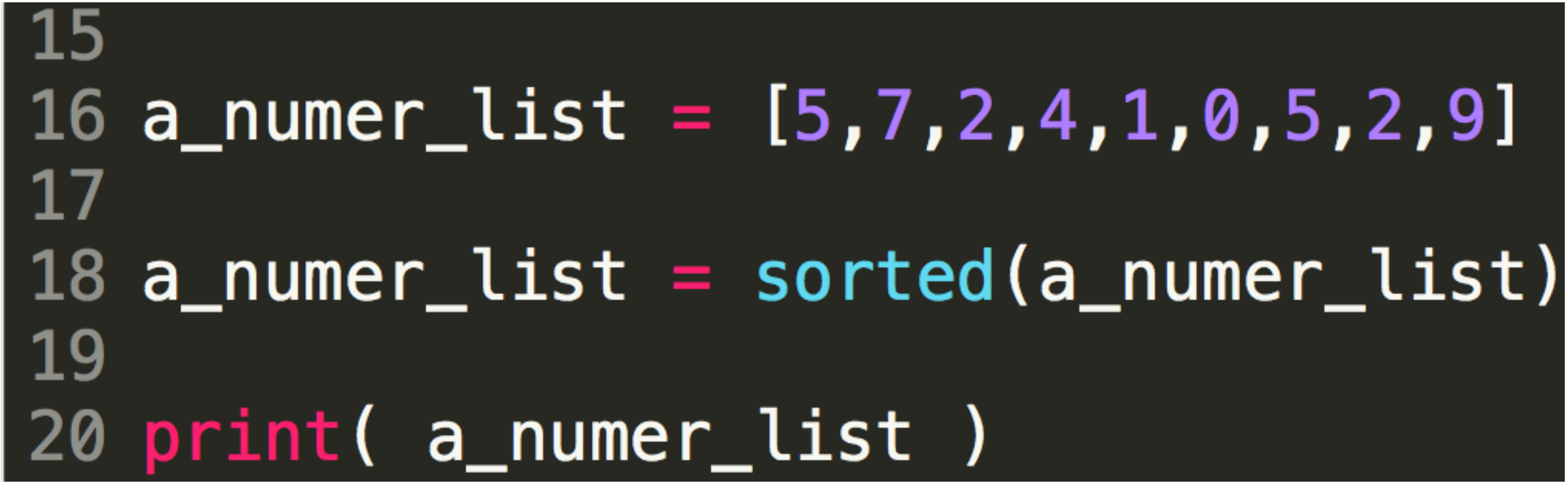
```python

a_number = 5

a_string = "hello"

a_number = str(a_number)

print(a_number + a_string)

```

Functions



```python
a_numer_list = [1,2,3,4,5,6,7,8,9]

print( len(a_numer_list) )
```

Functions

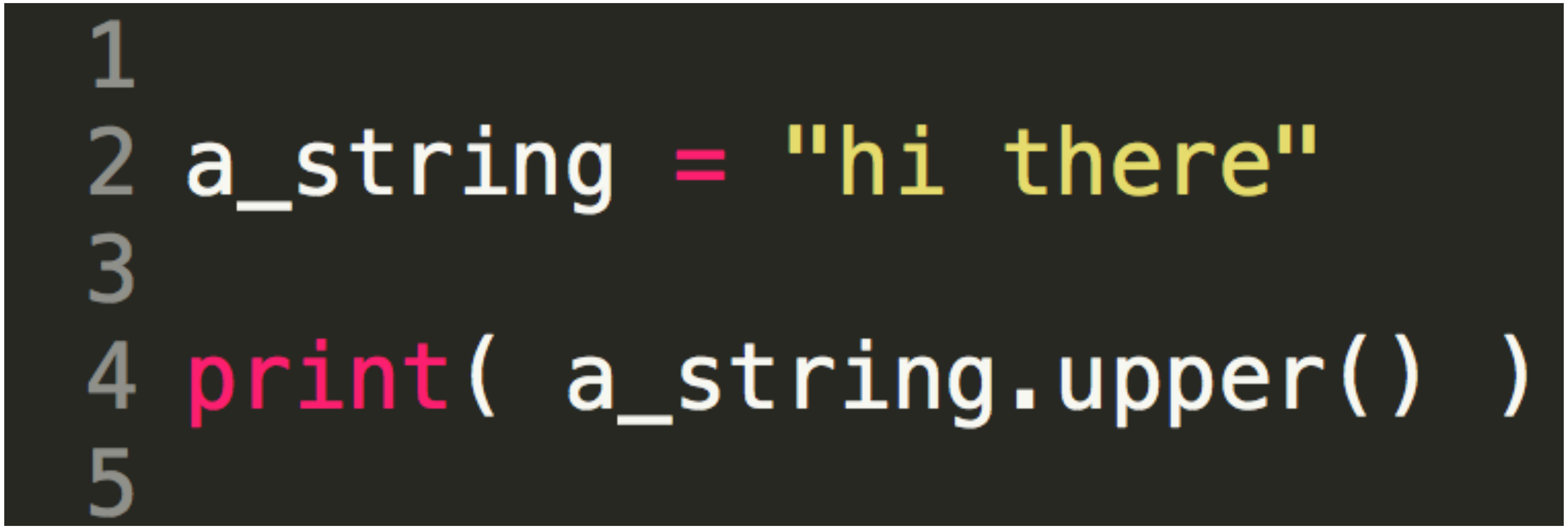


```python
a_numer_list = [5,7,2,4,1,0,5,2,9]

a_numer_list = sorted(a_numer_list)

print( a_numer_list )
```

Methods

- Methods are functions that exist as part of an object and uses that object's value as the parameter passed. Pretty much everything in python is an object and has methods

Methods



```

a_string = "hi there"

print( a_string.upper() )

```

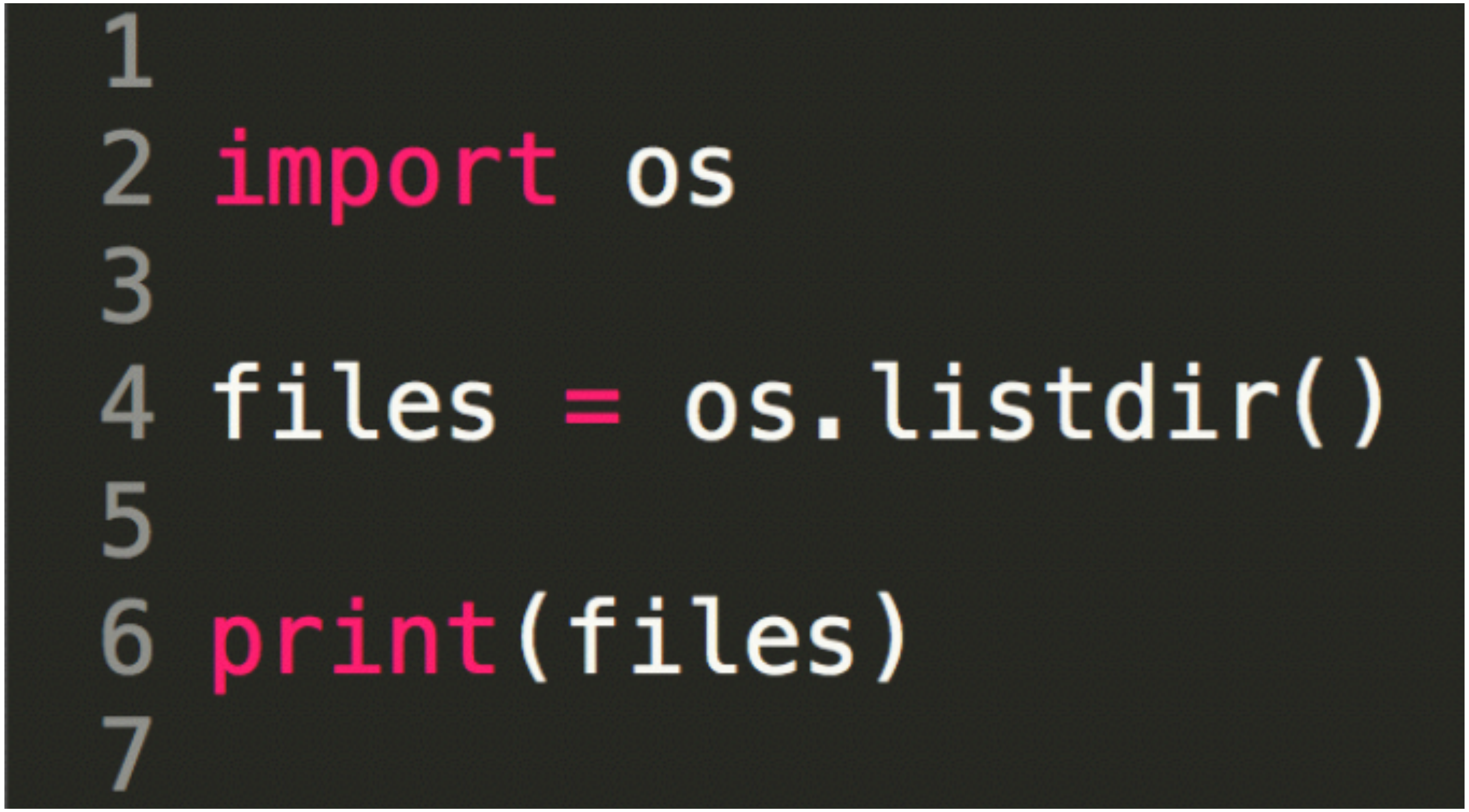
S.count(substring[, start[, end]])	Count occurrences of *substring*
S.endswith(suffix[, start[, end]])	True if *S* ends with *suffix*
S.find(substring [,start [,end]])	Find first occurrence of *substring* and return its index number; if not found, return -1
S.index(substring [,start [,end]])	Find first occurrence of *substring* and return its index number; if not found, raise ValueError
S.isalnum()	True if *S* has only alphanumeric characters
S.isalpha()	True if *S* has only alphabetic characters
S.isdigit()	True if *S* has only digits
S.isspace()	True if *S* has only whitespace characters
S.join(iterable)	Using *S* as a separator, stick together the strings in *iterable*
S.lower()	Convert *S* to lowercase
S.lstrip([chars])	Remove whitespace (or *chars*) from front (left) of*S*
S.replace (old, new[, count])	Replace *old* (a substring) with *new*
S.rfind(substring [,start [,end]])	Find the last (rightmost) occurrence of *substring*and return its index number; if not found, return -1
S.rindex(substring [,start [,end]])	Find the last (rightmost) occurrence of *substring*and returns its index number; if not found, raise ValueError
S.rstrip([chars])	Remove whitespace (or *chars*) from end (right) of *S*
S.split([separator [,maxsplit]])	Split *S* using whitespace (or *separator*) and return a list of substrings
S.startswith(prefix[, start[, end]])	True if *S* starts with *prefix*
S.strip([chars])	Remove characters at beginning and end of *S*; default is whitespace characters
S.upper()	Convert S to uppercase

Modules

- Modules add new functions to python beyond the built-in ones that are available all the time. But they need to declared at the start of the script so the processor knows it needs to include them.

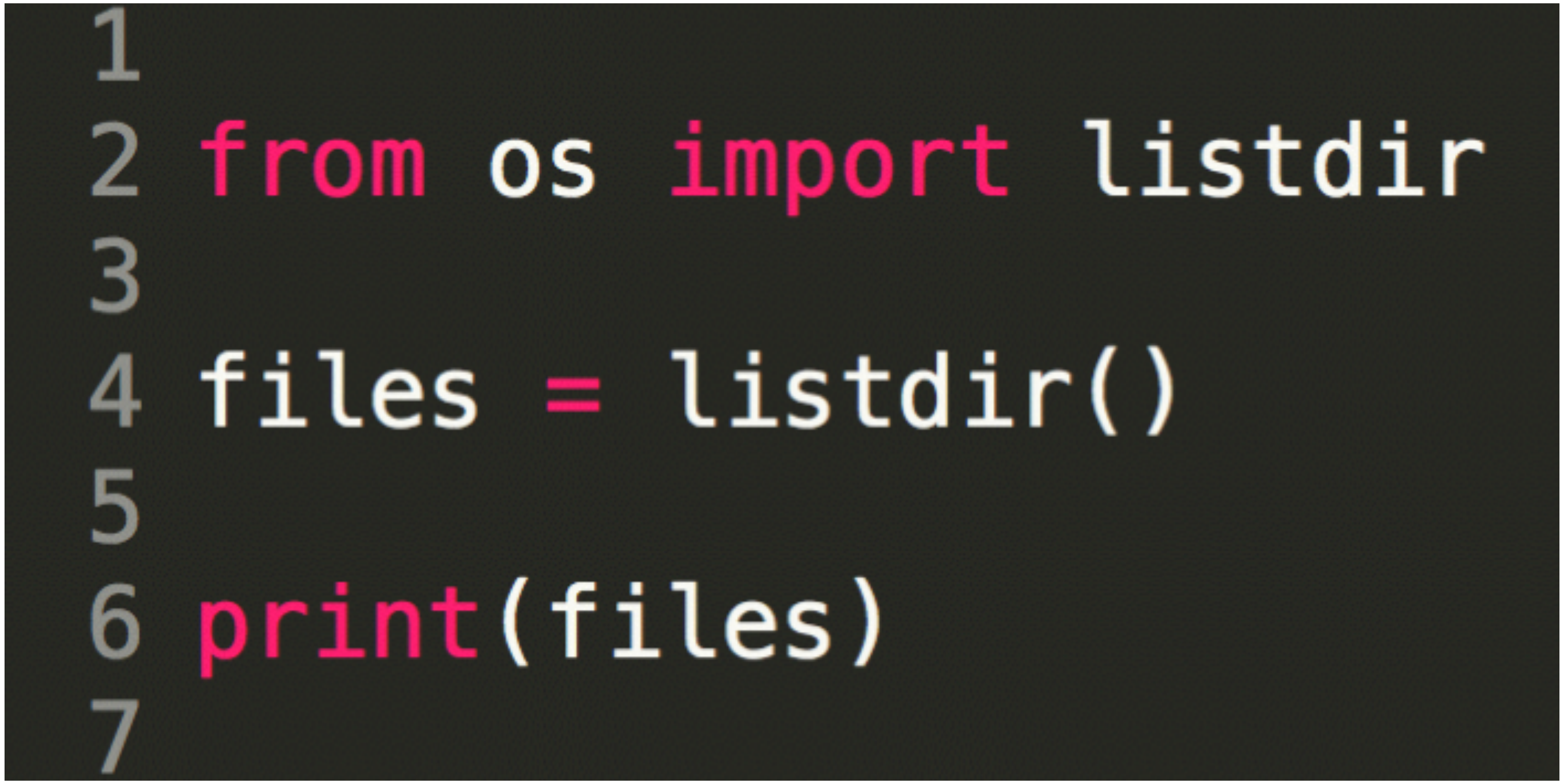
- Some modules are shipped with python by default and some are written by users that you can download and include in your projects.

Modules - os



```

import os

files = os.listdir()

print(files)

```

Modules - os



```python

from os import listdir

files = listdir()

print(files)

```

Challenge

- Let’s loop through three structures of increasingly complex data:
    - pratt_schedule1.py - A list
    - pratt_schedule2.py - A dictionary of lists
    - pratt_schedule3.py - A dictionary of dictionaries which hold lists.
- Loop through them successfully and print out specific information.