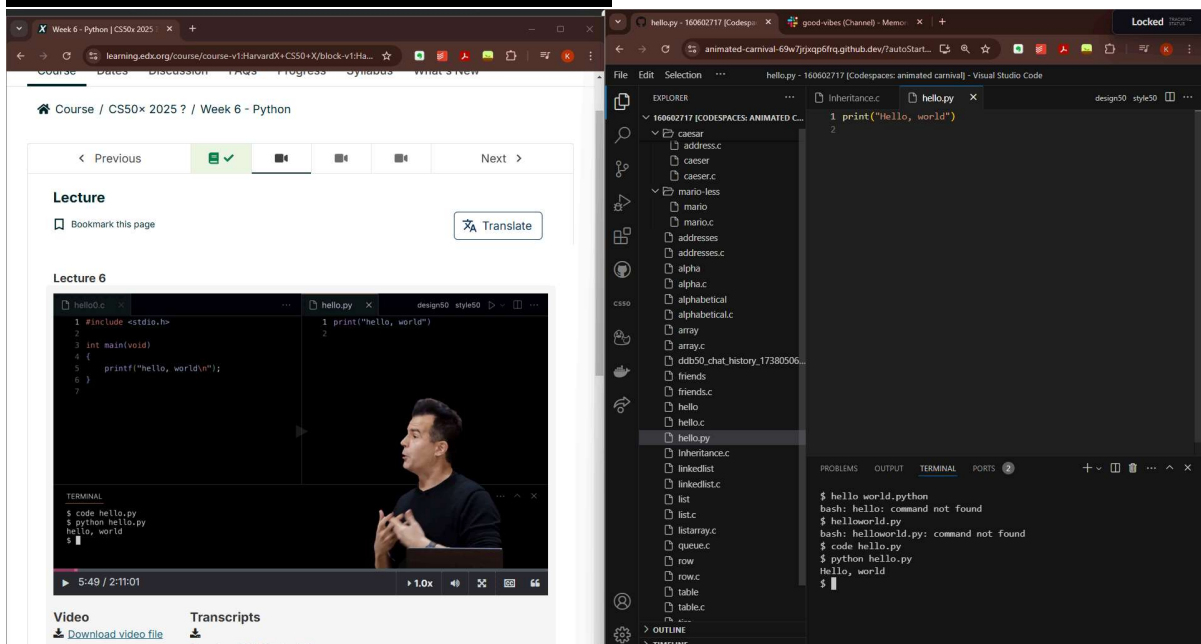


# CS50- Week-06: Python

## Lecture Notes

```
clang -o hello hello.c -lcs50
./hello
```

```
python hello.py
```



Course / CS50x 2025 ? / Week 6 - Python

Previous ☒ Next

Lecture

Bookmark this page [Translate](#)

Lecture 6

1 #include <stdio.h>  
2  
3 int main(void)  
4 {  
5 printf("hello, world\\n");  
6 }  
7

1 print("hello, world")  
2

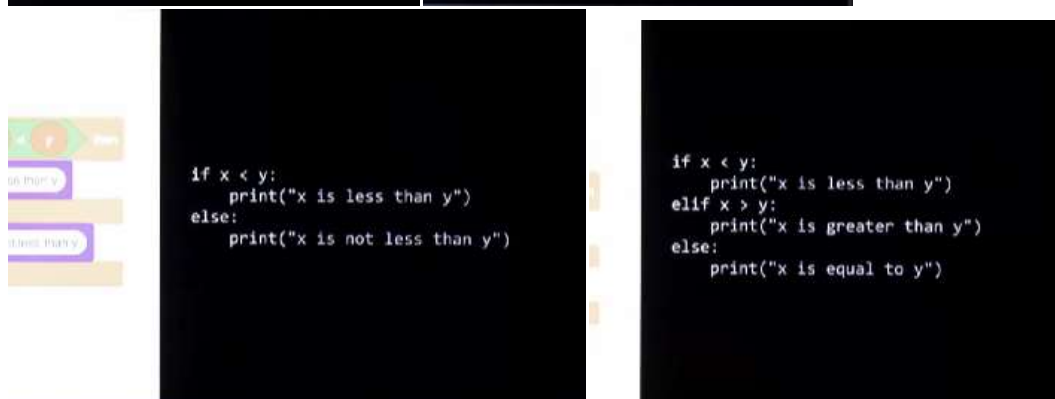
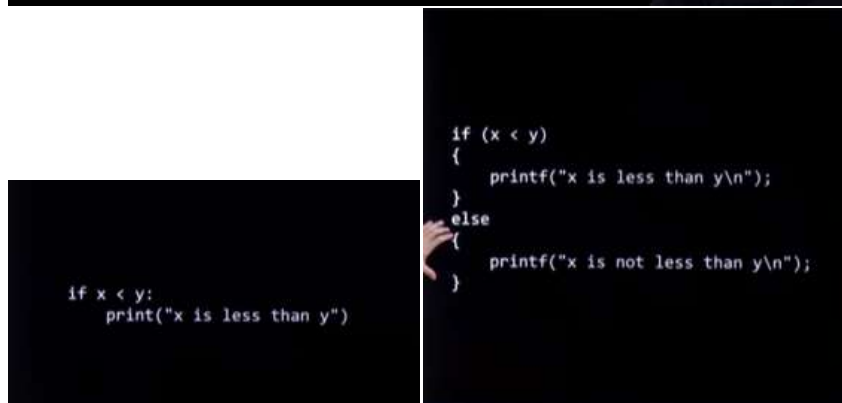
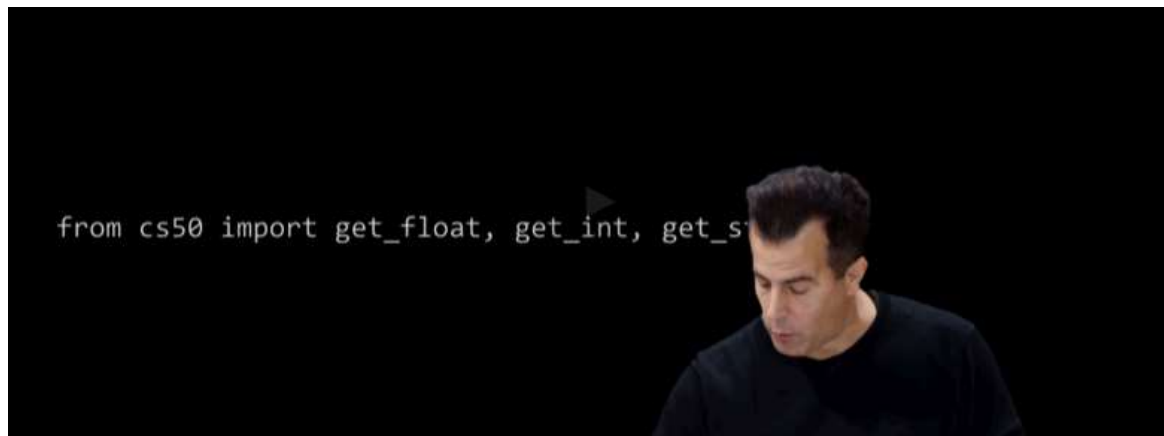
5:49 / 2:11:01

Video [Download video file](#) Transcripts

```
$ code hello.py  
$ python hello.py  
hello, world
```

```
$ hello world.python  
bash: hello: command not found  
$ helloworld.py  
bash: helloworld.py: command not found  
$ code hello.py  
$ python hello.py  
Hello, world  
$
```

```
from cs50 import get_float  
from cs50 import get_int  
from cs50 import get_string
```



Section-06: Yuliia;

# Agenda

- Syntax
- For Loops
- Dictionaries
- File I/O

```
char *phrase = get_string("...");
```

```
phrase = input("...")
```



```
if (strcmp(phrase, "hello") == 0)
{
    printf("Hi, %s!\n", name);
}
```

```
if phrase == "hello":
    print(f"Hi, {name}!")
```



```
my_list.append(3)
```

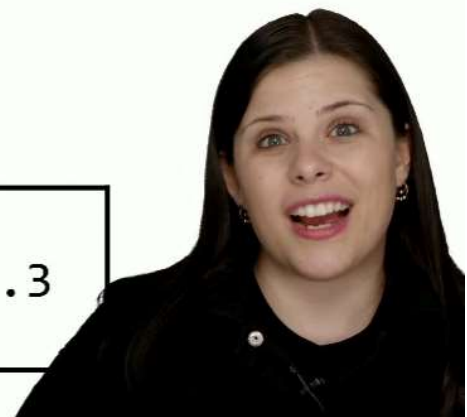
```
my_list
```

"Testing"	1	2.3	3
-----------	---	-----	---

```
my_list = ["Testing", 1, 2.3]
```

```
my_list
```

"Testing"	1	2.3
-----------	---	-----



```
my_list.append()  
    insert()  
    pop()  
    reverse()  
    sort()  
    ...
```



```
phrase = "You're off to Great Places"
```

```
phrase.capitalize()
```

```
phrase = "You're off to great places"
```

```
phrase.lower()  
    capitalize()  
    isspace()  
    split()  
    strip()  
    upper()  
    ...
```



```
for (int i = 0; i < 3; i++)  
{  
    print(i)  
}
```

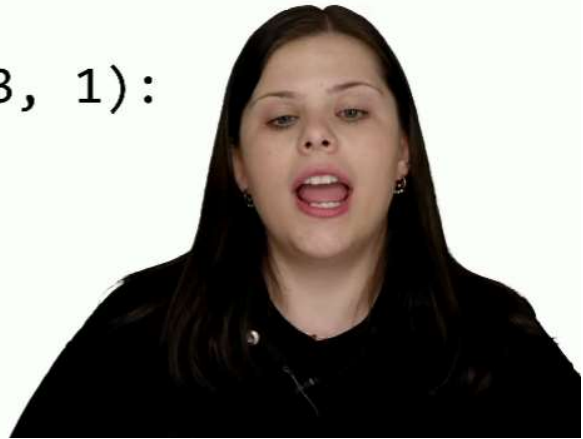


```
for i in [0, 1, 2]:  
    print(i)
```

Start (inclusive)



```
for i in range(0, 3, 1):  
    print(i)
```



Step



```
for i in range(0, 3, 1):  
    print(i)
```





# Dictionaries



```
song["name"]
```

song	
"name"	"Perfect"
"tempo"	95.05

songs[3]

```
songs =  
[{"name": "Perfect", "tempo": 95.0},  
{"name": "Eastside", "tempo": 89.0},  
{"name": "Wolves", "tempo": 124.9},  
{"name": "Him & I", "tempo": 87.0}]
```

```
with open(FILENAME) as file:  
    file_reader = csv.DictReader(file)
```

Shorts

## Python Syntax

- Python is an example of a very commonly-used modern programming language.
  - C was first released in 1972, Python in 1991.
- Python is an excellent and versatile language choice for making complex C operations much simpler.
  - String manipulation
  - Networking
- Fortunately, Python is heavily inspired by C (its primary interpreter, *Cpython*, is actually written in C) and so the syntax should be a shallow learning curve.

## Python Syntax

- To start writing Python, open up a file with the .py file extension.
- Unlike a C program, which typically has to be compiled before you can run it, a Python program can be run without explicitly compiling it first.
- Important note: In CS50, we teach **Python 3**. (Not Python 2, which is also still fairly popular.)

## Python Syntax

### • Variables

- Python variables have two big differences from C.
  - No type specifier.
  - Declared by initialization only.
  - Python statements needn't end with semicolons!

```
x = 54
```

## Python Syntax

### • Variables

- Python variables have two big differences from C.
  - No type specifier.
  - Declared by initialization only.

```
string phrase = "This is CS50";
```

## Python Syntax

### • Variables

- Python variables have two big differences from C.
  - No type specifier.
  - Declared by initialization only.

```
phrase = "This is CS50"
```

## Python Syntax

### • Conditionals

- All of the old favorites from C are still available for you to use, but they look a little bit different now.

```
if (y < 43 || z == 15)
{
    // code goes here
}
```

## Python Syntax

### • Conditionals

- All of the old favorites from C are still available for you to use, but they look a little bit different now.

```
if y < 43 or z == 15:
    # code goes here
```

## Python Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if y < 43 and z == 15:  
    # code block 1  
else:  
    # code block 2
```

## Python Syntax

- **Loops**

- Two varieties: while and for

```
counter = 0  
while counter < 100:  
    print(counter)  
    counter += 1
```

## Python Syntax

- **Lists**

- Declaring a list is pretty straightforward.

```
nums = [x for x in range(500)]
```

## Python Syntax

- **Lists**

- Tacking on to an existing list can be done a few ways:

```
nums = [1, 2, 3, 4]
nums.append(5)
```

## Python Syntax

- **Tuples**

- Python also has a data type that is not quite like anything comparable to C, a *tuple*.
- Tuples are ordered, immutable sets of data; they are great for associating collections of data, sort of like a struct in C, but where those values are unlikely to change.

## Python Syntax

- **Tuples**

```
presidents = [
    ("George Washington", 1789),
    ("John Adams", 1797),
    ("Thomas Jefferson", 1801),
    ("James Madison", 1809)
]
```

- This list is iterable as well:

```
for prez, year in presidents:
    print("In {1}, {0} took office".format(prez, year))
```

## Python Syntax

- **Tuples**

```
presidents = [  
    ("George Washington", 1789),  
    ("John Adams", 1797),  
    ("Thomas Jefferson", 1801),  
    ("James Madison", 1809)  
]
```

- This list is iterable as well:

```
for prez, year in presidents:  
    print("In {1}, {0} took office".format(prez, year))
```

```
In 1789, George Washington took office  
In 1797, John Adams took office  
In 1801, Thomas Jefferson took office  
In 1809, James Madison took office
```

## Python Syntax

- **Dictionaries**

```
pizzas = {  
    "cheese": 9,  
    "pepperoni": 10,  
    "vegetable": 11,  
    "buffalo chicken": 12  
}
```

## Python Syntax

- **Loops (redux)**

- The for loop in Python is extremely flexible!

```
for pie in pizzas:  
    # use pie in here as a stand-in for "i"
```

## Python Syntax

- Loops (redux)

```
pizzas = {  
    "cheese": 9,  
    "pepperoni": 10,  
    "vegetable": 11,  
    "buffalo chicken": 12  
}
```

```
for pie, price in pizzas.items():  
    print(price)
```

12  
10  
9  
11

## Python Syntax

- Loops (redux)

```
pizzas = {  
    "cheese": 9,  
    "pepperoni": 10,  
    "vegetable": 11,  
    "buffalo chicken": 12  
}
```

```
for pie, price in pizzas.items():  
    print("A whole {} pizza costs ${}".format(pie, price))
```

```
A whole buffalo chicken pizza costs $12  
A whole cheese pizza costs $9  
A whole vegetable pizza costs $11  
A whole pepperoni pizza costs $10
```

## Python Syntax

- Printing and variable interpolation

- format gives one way to interpolate variables into our printed statements in a very printf-like way, but there are others.

```
print("A whole {} pizza costs ${}".format(pie, price))  
  
print("A whole " + pie + " pizza costs $" + str(price))
```



## Python Syntax

### • Functions

- Python has support for functions as well. Like variables, we don't need to specify the return type of the function (because it doesn't matter), nor the data types of any parameters (ditto).

- All functions are introduced with the `def` keyword.

- Also, no need for `main`; the interpreter reads from top to bottom!

- If you wish to define `main` nonetheless (and you might want to!), you must at the very end of your code have:

```
if __name__ == "__main__":  
    main()
```

## Python Syntax

### • Functions

```
def square(x):  
    return x ** 2
```

## Python Syntax

### • Objects

```
struct car  
{  
    int year;  
    char *model;  
}
```

- C structures contain a number of *fields*, which we might also call *properties*.

- But the properties themselves can not ever stand on their own.

```
struct car herbie;  
year = 1963;  
model = "Beetle";
```



## Python Syntax

### • Objects

- You define a type of object using the `class` keyword in Python.
- Classes require an initialization function, also more-generally known as a *constructor*, which sets the starting values of the properties of the object.
- In defining each method of an object, `self` should be its first parameter, which stipulates on what object the method is called.

## Python Syntax

### • Objects

```
class Student():  
  
    def __init__(self, name, id):  
        self.name = name  
        self.id = id  
  
    def changeID(self, id):  
        self.id = id  
  
    def print(self):  
        print("{} - {}".format(self.name, self.id))
```

## Python Syntax

- **Style**

- If you haven't noticed, good style is **crucial** in Python.
- Tabs and indentation actually matter in this language, and things will not work the way you intend for them to if you disregard styling!
- Good news? No more curly braces to delineate blocks!
  - Now they just are used to declare dictionaries.

## Python Syntax

- **Including files**

- Just like C programs can consist of multiple files to form a single program, so can Python programs tie files together.

```
cs50.get_int()  
cs50.get_float()  
cs50.get_string()
```