An Introduction to Python

Part I: Learning the Basics

Introduction

- **Mentors:** Ian Leinbach & Tyler Kugler
- **Overview of the program**
  - 5 sessions designed for Python and computer science beginners
  - **Goals:**
    - Learn the basics of Python programming
    - Learn fundamental data structures and algorithms
- **Structure of the session**
  - Lecture including small exercises for each topic
  - Activities at the end to review what we learned

Lecture 1 Overview

1. **Introducing the Terminal**
2. **Running Python Programs**
3. **How to Code: The Building Blocks**
    a. **Essentials:** data types, operators, variables, expressions, statements, and functions
    b. Built-in functions
4. **Flow Control**

General Overview

- **What is computer science?**
- **What is a program?**
- **What is Python?**

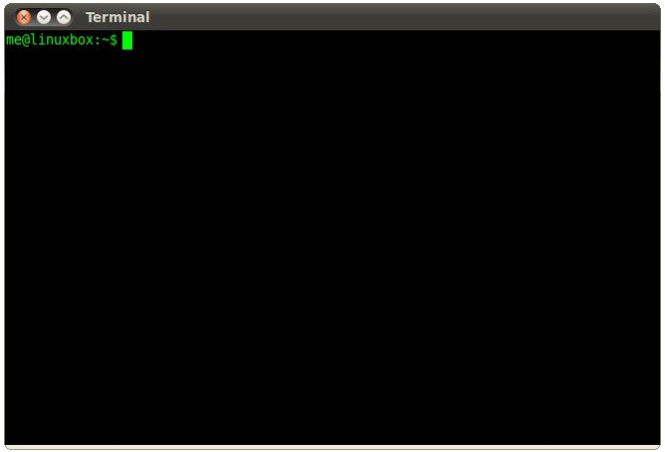
python™

First, the Terminal

We need a way to access the computer to run our programs, so we use the terminal to talk directly to the computer:

- To navigate our file system
- To run commands and programs

DANGER: There's a one line command that can delete the entire computer, so please don't run that command!



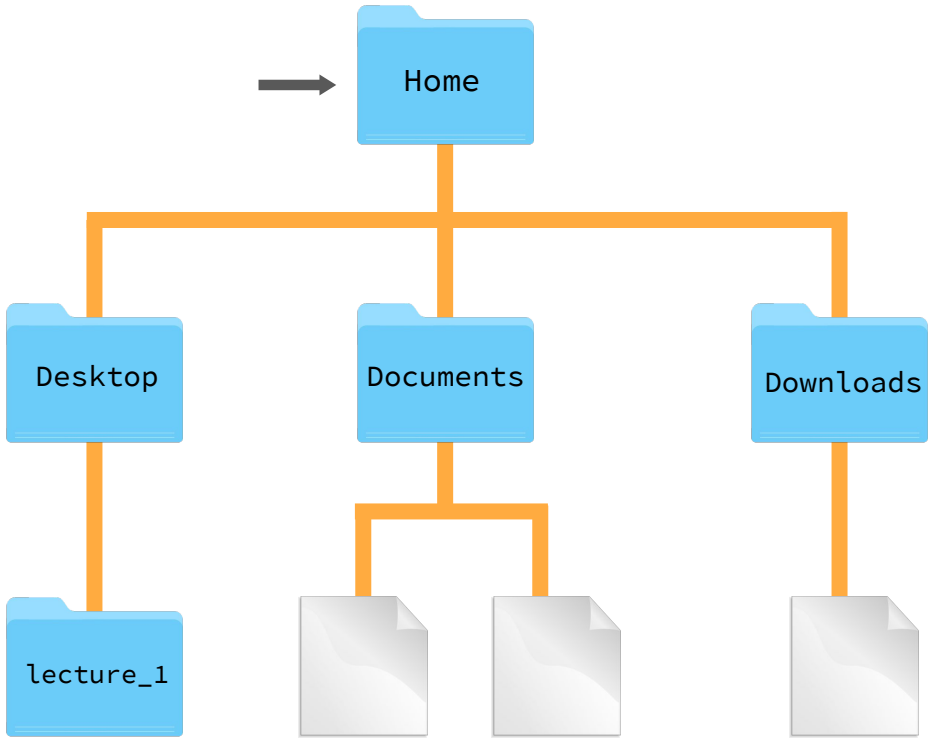
Getting Started on the Terminal

Some useful terminal commands:

cd [x]	**c**hange **d**irectory (folder) to [x] to move up a level, enter **“cd ..”**
mkdir [x]	**m**ake a **d**irectory named [x]
ls	Lists all of the files in the current directory
python3	opens up a Python interpreter in the terminal
quit()	closes an open python interpreter

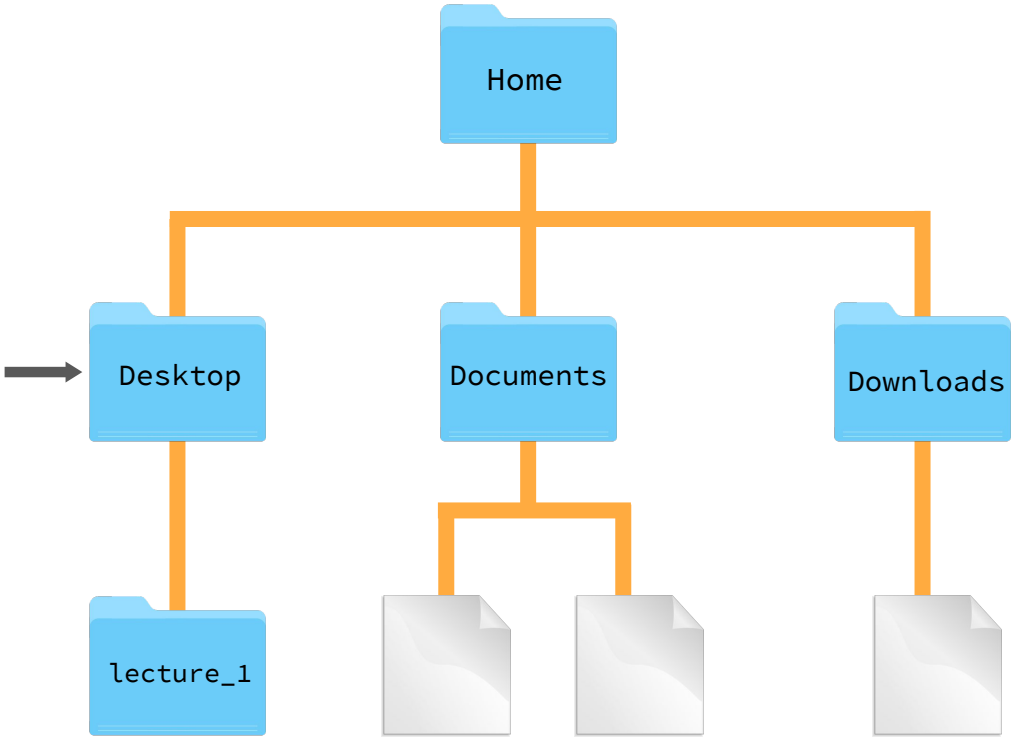
---

ctrl + c	in case of emergencies
tab	autocompletes the command


Working Directory
Home
Desktop
Documents
Downloads
lecture_1

```
cd Desktop
```

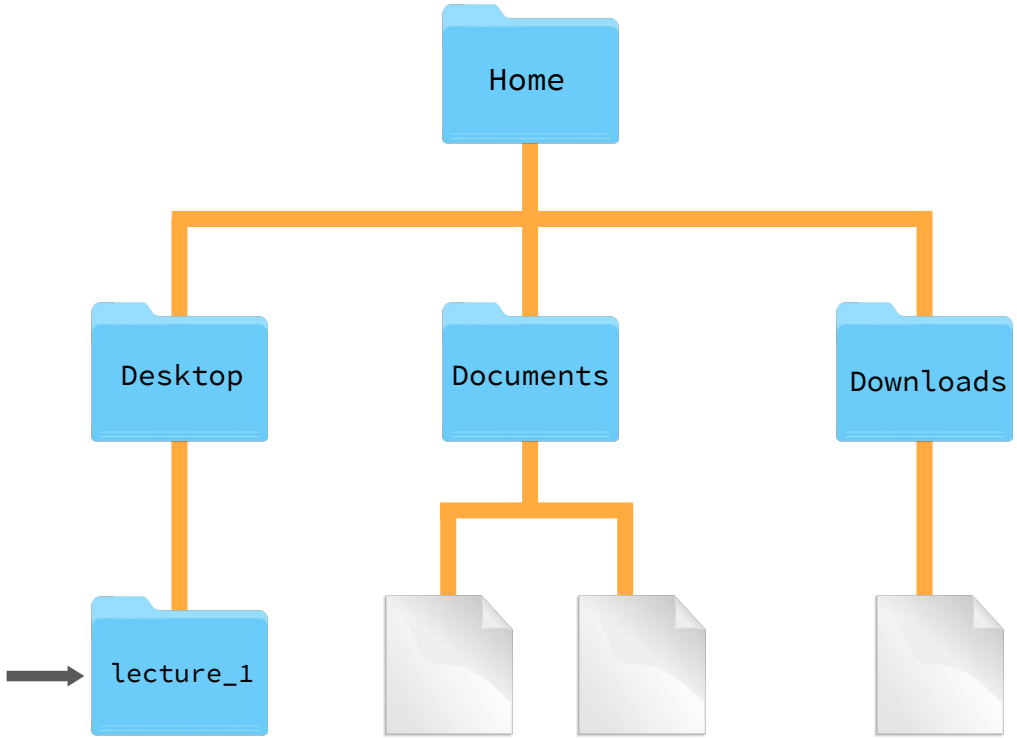
Working Directory



```
cd Desktop
```

```
cd lecture_1
```

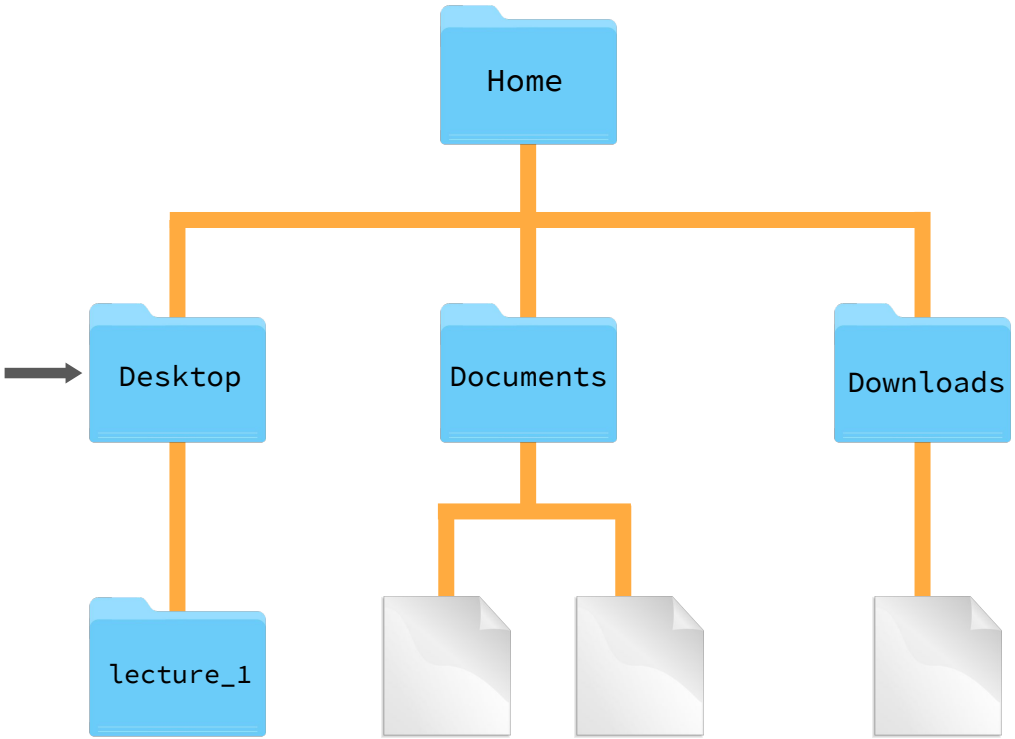
Working Directory



```
cd Desktop
```

```
cd lecture_1
```

```
cd ../
```

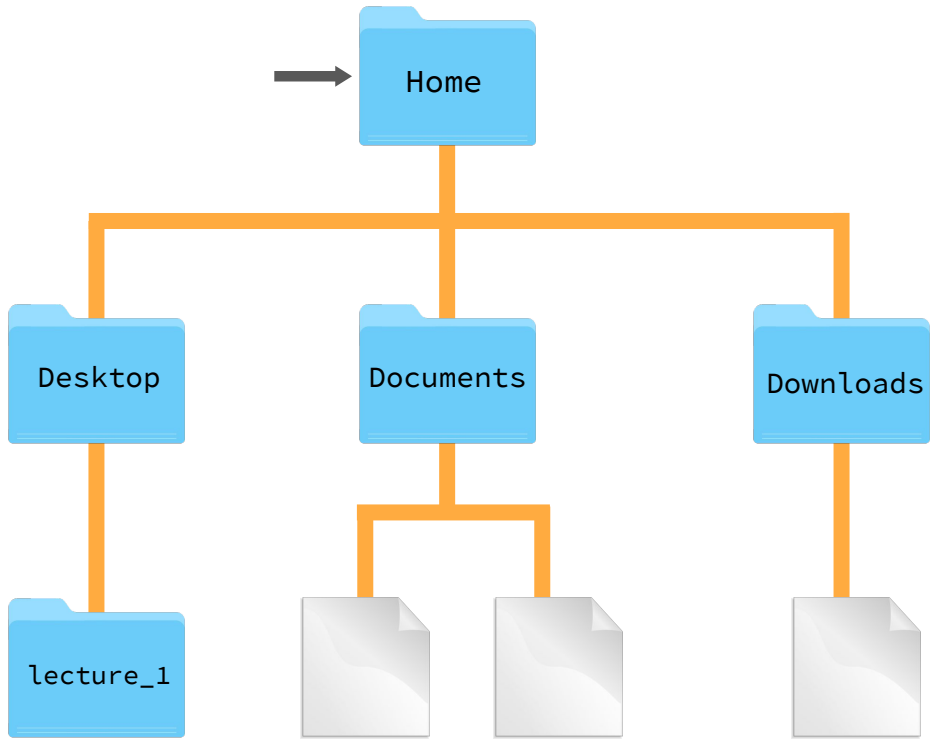


```
cd Desktop
```

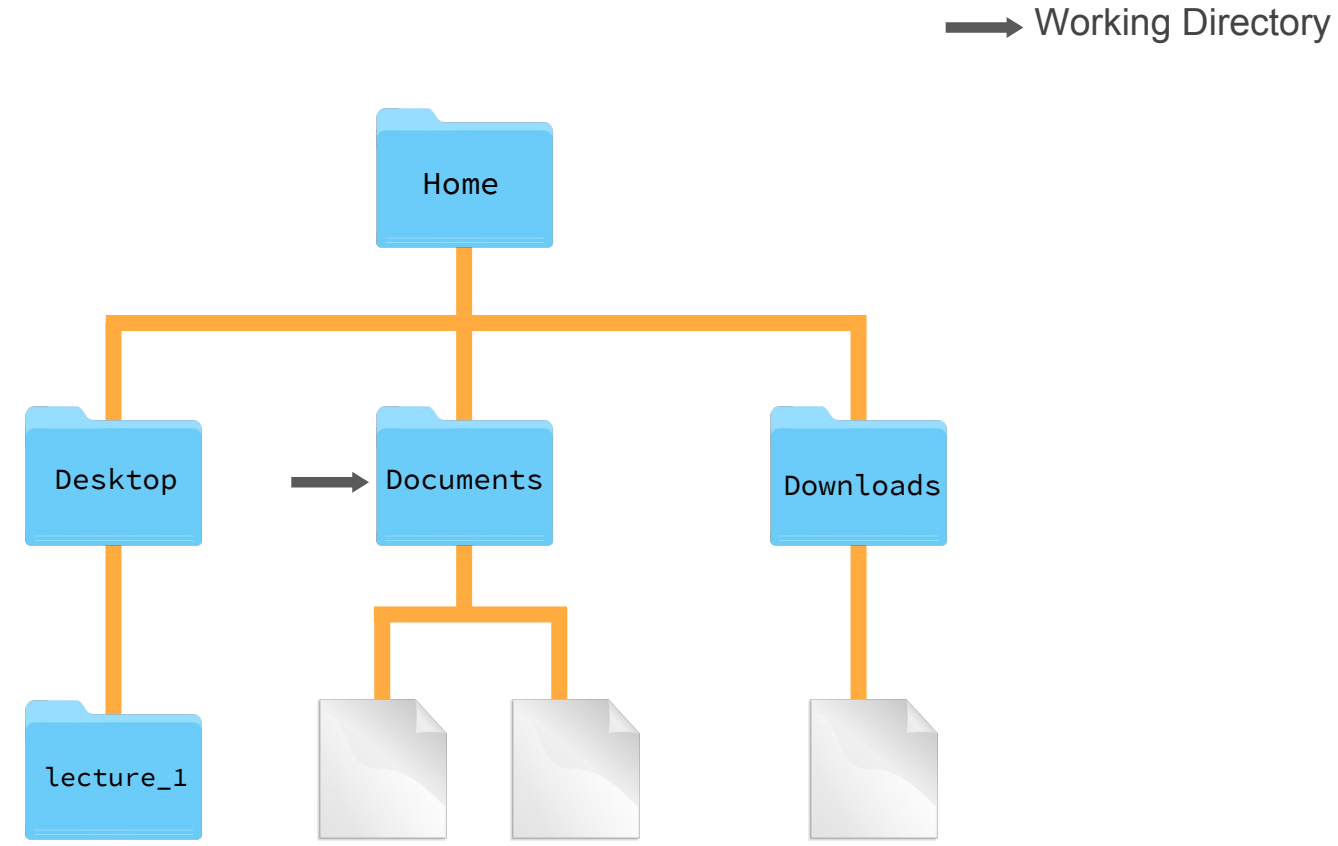
```
cd lecture_1
```

```
cd ../
```

```
cd ../
```



```
cd Desktop
cd lecture_1
cd ../
cd ../
cd Document
```



Navigating Files and Python

Try it out!

1. Change to the Desktop directory: `cd Desktop`
2. Create a directory called **lecture_1:** `mkdir lecture_1`
3. List all of the files in the current directory `ls`
4. Change to the **lecture_1** directory: `cd lecture_1`
5. Open a Python interpreter: `python3`
6. Enter a mathematical expression: `5*4`
7. Close the Python interpreter `quit()`

Commands: cd, mkdir, ls, python3, quit()

Running Python Programs

Python’s Interactive Console

- By typing in **python3** into the terminal, we open up an interactive mode called the **console** where we can type python commands and press enter to immediately see the results.

Pros	Cons
You get to play around with Python quickly	The commands you type in don’t get saved for later

Console vs. File

- There are multiple ways to run Python code
  - The console is interactive, but the code doesn't get saved

- **Problem:** programs are often comprised of many, many lines
- **Solution:** write your code in a .py file, and run that file on the terminal

```
$ python3 myFile.py
```

- We will experiment with both methods in this session!

Run a Python File

Try it out!

First, we have to get a file:

1. Download the file **`welcome.py`** from the Google Drive Folder
2. Click the little arrow and select “Show in folder”
3. Drag and drop the file into your **`lecture_1`** folder on the Desktop

Now let’s run it:

1. Type: **`python3 welcome.py`** that’s it!
2. Press: **`ctrl + c`** this will end the program

How to Code: The Building Blocks

Data Types in Python

1. **Strings**
   a. “things in quotations”
2. **Integers**
   a. 1, 2, 3, 4...
3. **Floats**
   a. 3.14159
4. **Booleans**
   a. True
   b. False

Operators

Operators let us manipulate and evaluate our data.

+ − * / < > <= >=

They behave differently depending on what data type they're operating on:

- 1 + 1 → 2
- "hello " + "world" → "hello world"

Operator Weirdness

Sometimes, you can mix and match data types:

“hello” * 3 works, **but** “hello” - 3 doesn’t work

- 1 + 1 → 2, **but** “1” + “1” → “11”
- “1” + 1 → **ERROR** (you can’t add strings to integers)
- “Hello” * 3 → “HelloHelloHello”
- 2 ** 3 → 8

Strings

```
“This is a string”
```

```
‘this is also a string’
```

```
“$0 1$ +H1$”
```

Strings

Try it out!

1. Type: `"I'm a string."` — Seems simple enough
2. Type: `"I'm a sentence." + "Same?"` — Spaces matter!
3. Type: `'"Quotes" are no problem.'` — Swap them if you want

What if I only want some of a string?

4. Use brackets: `"abcdef"[0]` — Strings are 0-indexed
5. More than one: `"abcdef"[2:4]` — Gets you 2 → 4 ***not*** including 4
6. Now try: `"abcdefg"[2:]` — How convenient!

If `[2:]` works… Any ideas what else might...?

Variables

We can name our data to make our code more dynamic

`>>> str = “hi”`	This assigns the value “hi” to str
`>>> str` `“hi”`	We can check what’s being stored in str. But what if we want to change what is being stored there. We know how to concatenate!
`>>> str + “ there!”` `“hi there!”`	Okay, seems good. We even remembered a space! Let’s just check str again to be sure.
`>>> str`	What do you get?

Expressions

A sequence of *commonly typed* values or other expressions. We can also throw in some functions and operators to relate all the values in our sequence. Expressions are simplified by Python to an equivalent, single value.

```
>>> 35 - 31 == 4
True
```

```
>>> “abcd\nefgh”
“abcd\nefgh”
```

Statements

A unit of instruction or action that tells Python something to do. If you ask, “What is this equivalent to?” and it doesn’t really make sense, it’s probably a statement.

```
>>> x = 3
```

```
>>> print(“abcd\nefgh”)
abcd
efgh
```

Let's Revisit Variables

Try it out!

1. Type again: `str = "hi"` — Okay, we've seen this
2. And again: `str` — Still good
3. Like last time: `str + " there!"` — Wait a second!

That's an expression! Expressions don't change anything because they are only representations of data. We need a statement to change something about the *state* of the program.

4. How about: `str = str + " there!"`

The expression we used before is what we want str to hold, but we have to instruct Python to make a change. We use the assignment operator and the desired expression together!

5. Now type: `str` — That checks out, nice!

Functions

A function takes in input and then returns an output after performing operations on the original input.

***Note:** not all functions in programming return an output, sometimes they just perform a procedure, like the `print()` function.

Useful, Built-in Functions

print(string/number)

Takes any number of arguments, prints it to the screen

```
>>> print(“hello, world!”)
hello, world!
```

But wait, why not just type the string by itself like before?

```
>>> “hello, world!”
“hello, world!”
```

So you’re telling me **print()** just gets rid of quotes?

```
>>> print(“hello,\n world!”)
hello,
 world!
```

print() doesn’t return anything. Instead, it performs a process: formatting the string and displaying it.

input(string)

```
>>> name = input(“Enter your name: ”)
Enter your name: Grace Hopper
>>> name
“Grace Hopper”
```

input() prompts the user for input and stores it in a variable.

```
>>> age = input(“Enter your age: ”)
Enter your name: 18
>>> age
“18”
```

Notice that **input()** always puts a string into the variable.

len(string)

Returns the length of a string

```
>>> len(“some text”)
9
```

```
>>> str = “a string”
>>> len(str)
8
```

str.**replace**(old, new)

Return copy of **str** with the first instance of **old** replaced by **new**

```
>>> word_one = “this is a string”
>>> word_two = word_one.replace(“hi”, “BYE”)
>>> print(word_one)
this is a string

>>> print(word_two)
tBYEs is a string
```

aside: **Comments**

```
# anything written after a '#'
# will be ignored by Python
#
#
#
```

This is super useful for documenting what your code does!

Exercise

Try it yourself!

1. Create a new file called **`hello_world.py`** by typing:

   ```
   idle3 hello_world.py
   ```

2. Write the following program in the file:
   - Print out “Hello, world!”
   - Create a variable called **name** to store input
   - Print out “Hello, [**name**]!”
   - Print out how many letters are in their name
3. Quit out of the console by entering: **`quit()`**
4. Run the file in terminal: **`python3 hello_world.py`**

Control Flow

What is Control Flow?

- Usually programs are ran line-by-line
- But what if you don’t want to run a certain line of code if some condition is true?

For example:

```
if test is tomorrow:
    study()

otherwise:
    party()
```

We want to test that a certain **condition** is true, and then execute code in a different way to make our programs smarter

Conditionals / Boolean Expressions

- Conditionals evaluate to either **True** or **False**

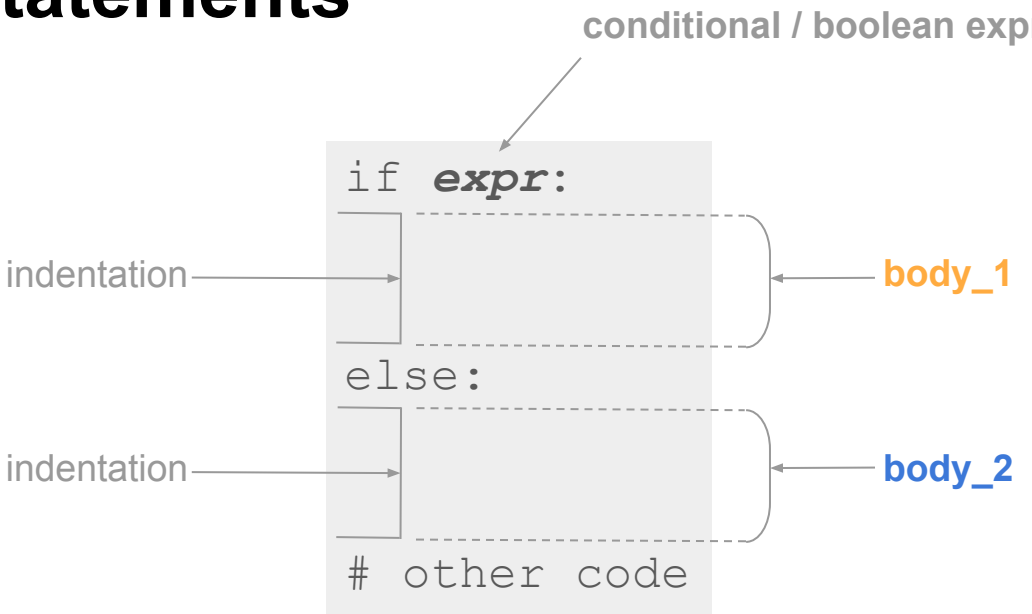
  2 > 1 → **True** 2 < 1 → **False** 1 == 1 → **True** 1 != 1 → **False**

  “abc” < “xyz” → **True** “abc” > “xyz” → **False**

  You can combine expressions with the **and** as well as **or** operators:

  - (2 > 1) **and** (1 < 2) → **True** because expr_1 *and* expr_2 are both true.
    - (5 > 10) **and** (2 < 3) → **False** because only one of them evaluates to true.
  - (2 > 1) **or** (1 > 3) → **True** because expr_1 *or* expr_2 is true.
    - (1 < 0) **or** (5 > 10) → **False** because neither evaluates to true.

If-else Statements

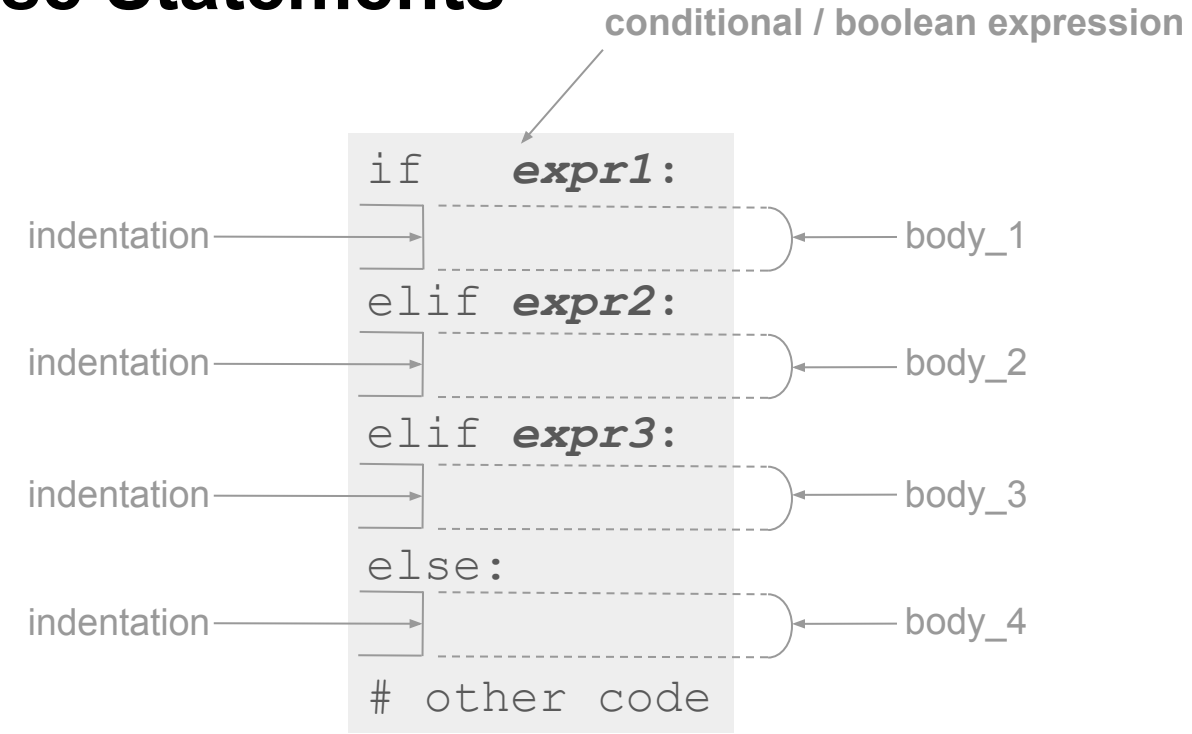


- ***expr*** evaluates to **true** → execute **body_1** → move on to other code
- ***expr*** evaluates to **false** → execute **body_2** → move on to other code

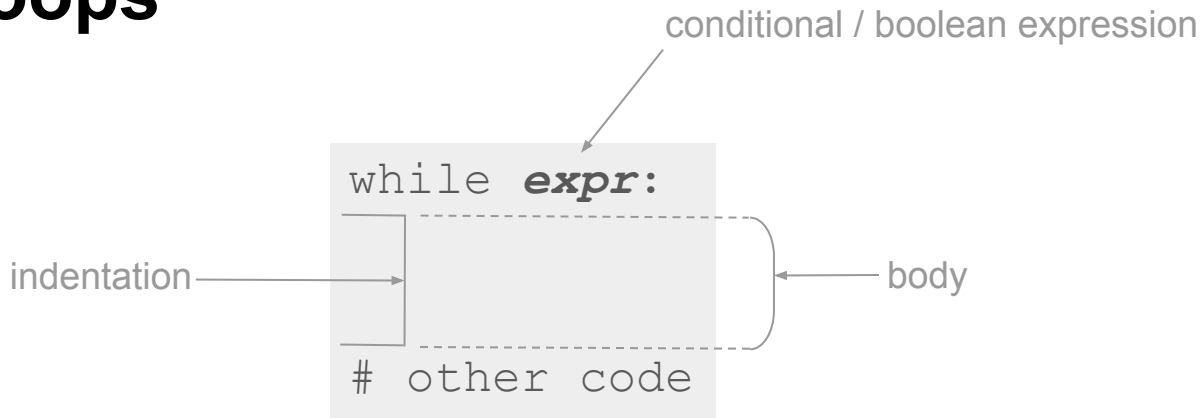
If-else Statements: Example

```
if len(“Google”) > 5:
    print(“Hello!”)
else:
    print(“What’s up?”)
```

If-elif-else Statements



While Loops



expr evaluates to **true** → execute body until ***expr*** evaluates to **false**
expr evaluates to **false** → move on to other code

Exercise 3

Try it yourself!

1. Create a variable and set it to 0
2. Create a while loop
    - The conditional should check if your variable is less than 10
    - The body should print out the current value of your variable, and then add 1 to it
3. Once your while loop is working, try adding an if-statement
    - Modify the body of the while loop so that it only prints out the value of the variable when the value *isn't* equal to 5.

```
while expr:
```

Summary

- **The Terminal**
  - How to talk directly to your computer (introduced common terminal commands)
- **Running Python**
  - Interactively on the console
  - Longer programs written and saved in files
- **Building Blocks of Writing Code**
  - Data Types
  - Variables, Expressions, and Statements
  - Built-in Functions
  - Comments are important!
- **Control Flow**
  - If-else Statements
  - If-elif-else Statements
  - While Loops