Python Crash Course

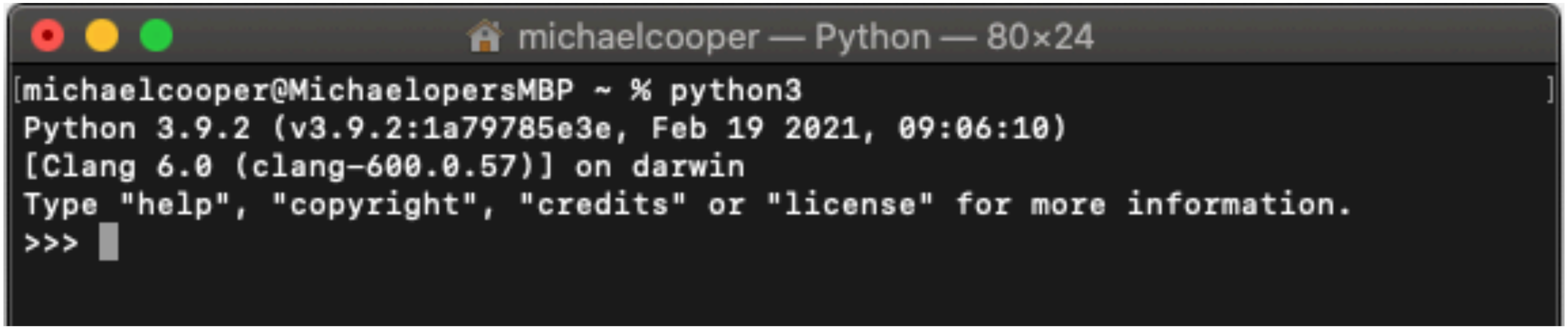
Python Crash Course

- Interactive Interpreter
- Variables and Types
- Numbers and Booleans
- Strings and Lists
- String Formatting
- Control Flow
- Loops
- Functions

Interactive Interpreter

Python is an *interpreted language*.

Demo: Interactive Interpreter



Follow along with the lecture using your own interactive interpreter!

Interactive Interpreter: Procedure

1. Parse Python code written at the prompt.
2. Evaluate the code.
3. Display the result.

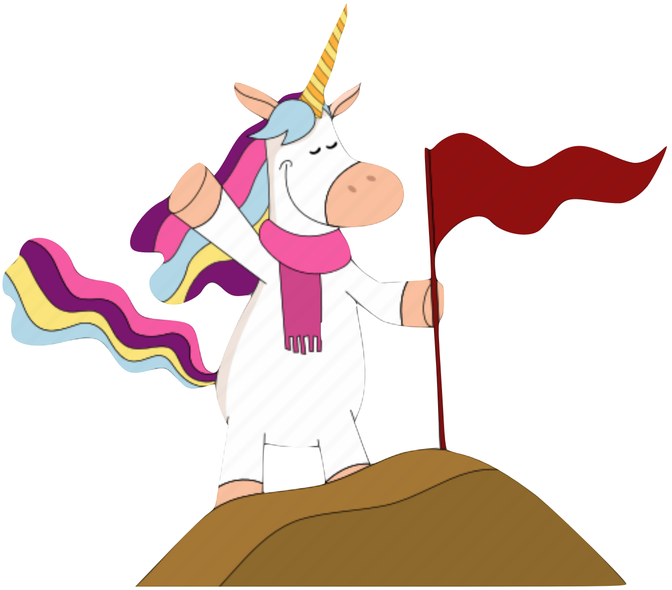
Interactive Interpreter: Summary

- Sandbox to experiment with Python.
- Shortens code-test-debug cycle.
- One of the greatest things ever.

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- Variables and Types
- Numbers and Booleans
- Strings and Lists
- String Formatting
- Control Flow
- Loops
- Functions



Variables and Types

What's Your Type?

Today, we'll explore the following object types:

- Numeric Types: `int`, `float`
- Sequence Types: `string`, `list`

Variables and Types

Variable declaration in C/C++/Java:

```
int x = 5;
```

Type Name Value

Variable declaration in Python:

```
int x = 5;
```

Type Name Value

Python automatically infers the type from the value.

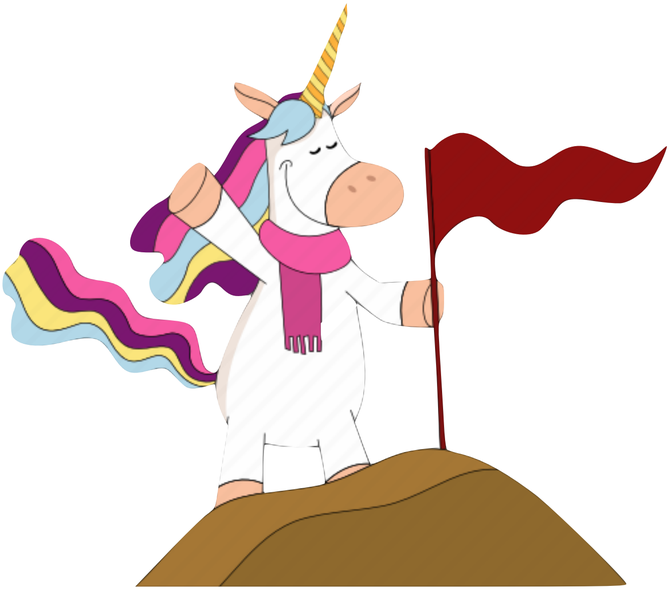
Variables and Types: Summary

- Python variables are dynamically typed: they take on the type of the object they are representing.
- Variables can change value *and type* over the course of program execution. Variables, therefore, are not explicitly bound by the programmer to a type when they are defined.
- Type of a variable `var` can be checked by calling `type(var)`.

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- Numbers and Booleans
- Strings and Lists
- String Formatting
- Control Flow
- Loops
- Functions



Numbers and Booleans

Numbers and Booleans: Summary

Numeric Operation	Operation Syntax	Assignment Syntax
Addition	`x + 5`	`x += 5`
Subtraction	`x - 5`	`x -= 5`
Multiplication	`x * 5`	`x *= 5`
Division	`x / 5`	`x /= 5`
Integer Division	`x // 5`	`x //= 5`
Modulo Operator	`x % 5`	`x %= 5`
Exponentiation	`x ** 5`	`x **= 5`

Boolean Operation	Operation Syntax
Not	`not a`
And	`a and b`
Or	`a or b`
Equals (Not Equals)	`a == b (a != b)`
Greater (Or Equal)	`a > b (a >= b)`
Less (Or Equal)	`a < b (a <= b)`
Chained Expressions	`a > b > c`

Numbers and Booleans: Overview

- Python supports several numeric (`int`, `float`) types, and the `bool`ean type. Python supports various atomic operations on them.
- Python makes assumptions about these types which may differ from other languages (division, for example, is not integer division by default).

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- Strings and Lists
- String Formatting
- Control Flow
- Loops
- Functions

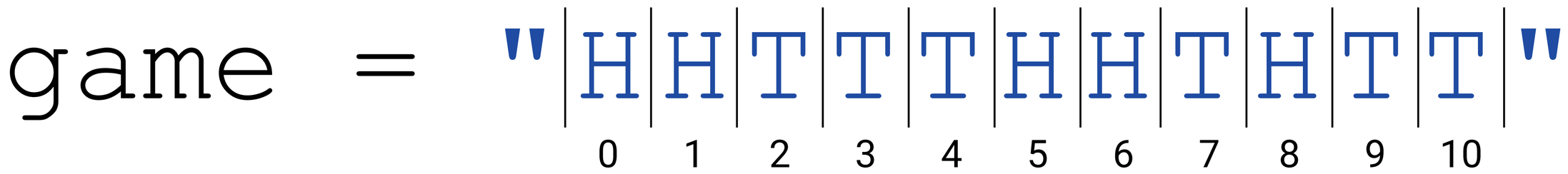
Strings and Lists

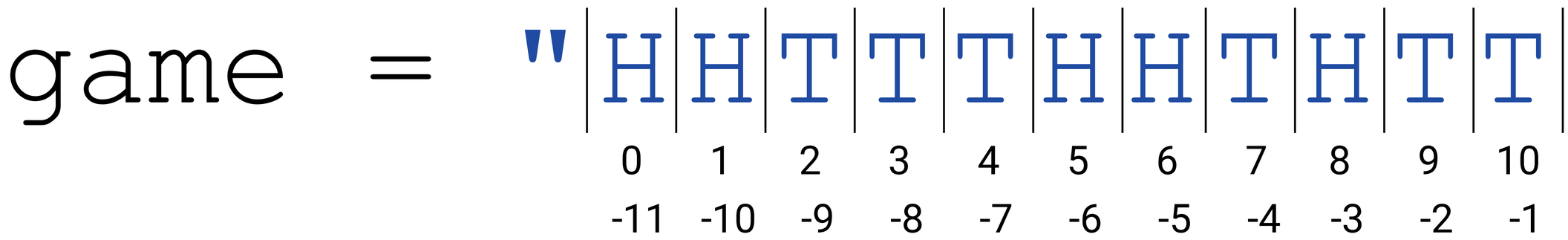
```
game = "HHTTTHHHTHTT"
```

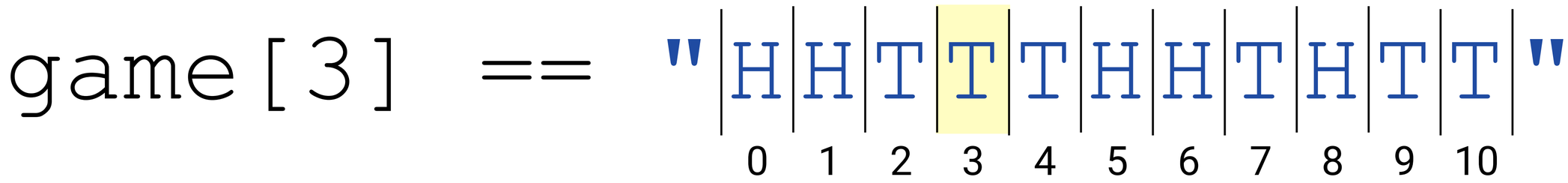
```
len(game)
```

"HTH" in game

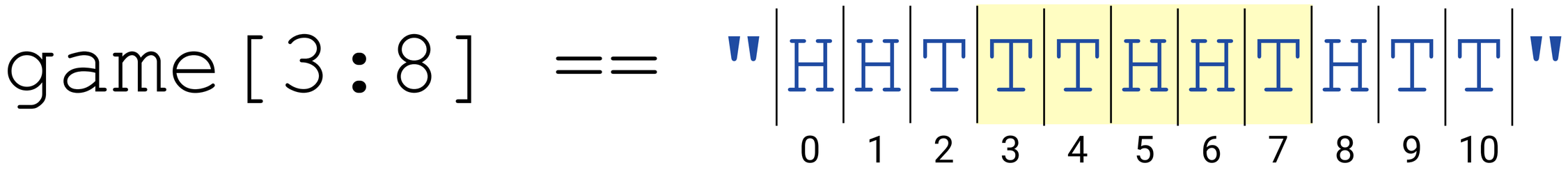
```
game = "HHTTTHHHTHTT"
```

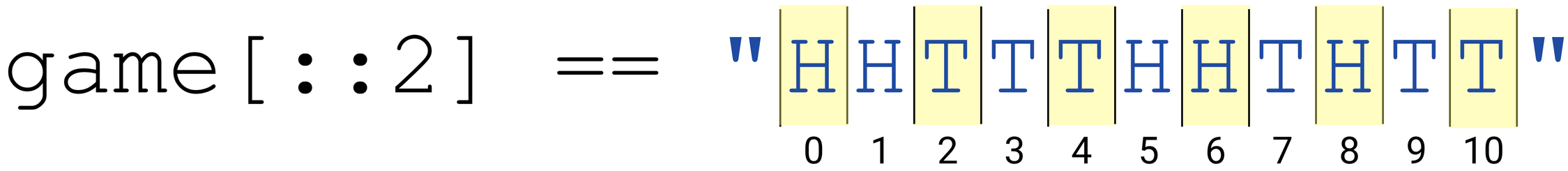

game = "|H|H|T|T|T|H|H|T|H|T|T|
0 1 2 3 4 5 6 7 8 9 10


game = "|H|H|T|T|T|H|H|T|H|T|T|
0 1 2 3 4 5 6 7 8 9 10
-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1


game[3] == "|H|H|T|T|T|H|H|T|H|T|T|
0 1 2 3 4 5 6 7 8 9 10


game[-2] == "|H|H|T|T|T|H|H|T|H|T|T|
-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1


game[3:8] == "|H|H|T|T|T|H|H|T|H|T|T|
0 1 2 3 4 5 6 7 8 9 10


game[::2] == "HHTTTHHTHTT"
0 1 2 3 4 5 6 7 8 9 10

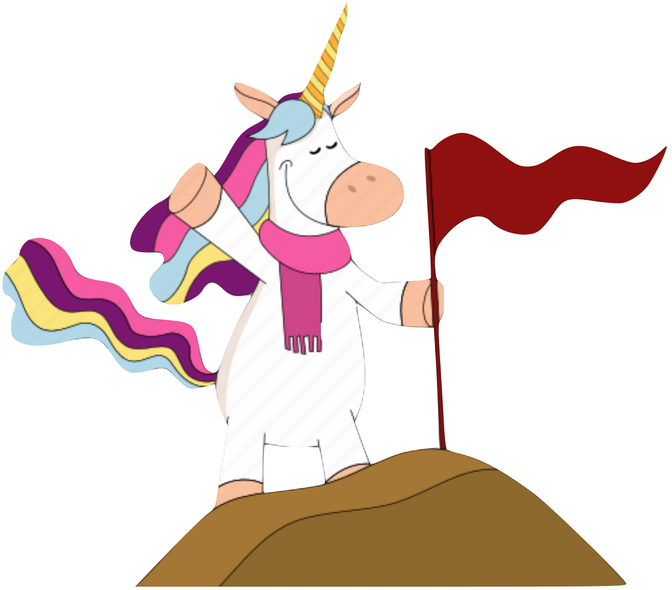
Strings and Lists: Summary

- Python supports strings and lists as primitive types. Lists can store multiple elements of different types.
- Strings and lists can be indexed into and sliced. Slicing notation is of the form `my_str[start:end:step]`. Indices, and step sizes, can be either positive or negative.
- Python supports membership queries over lists using the `in` keyword.

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- ~~Strings and Lists~~
- String Formatting
- Control Flow
- Loops
- Functions



String Formatting

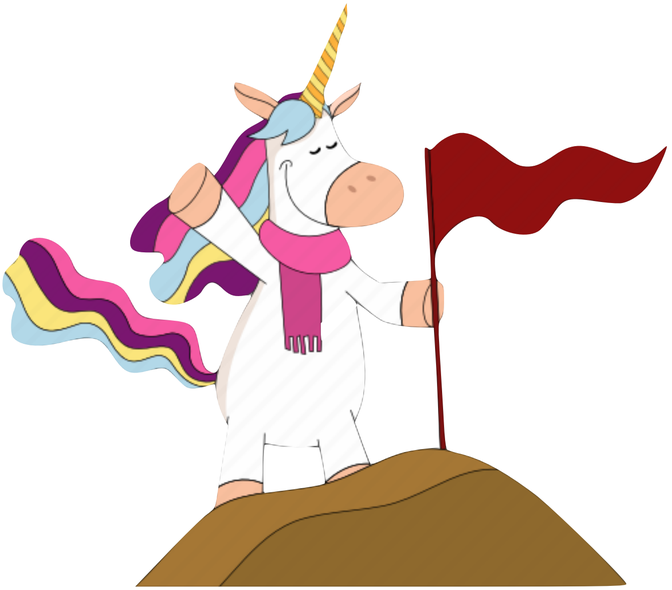
String Formatting: Summary

- String formatting refers to the process of constructing a string using the values of variables in code. Python supports several techniques of string formatting.
- Both the `.format` method and f-strings automate much of string formatting. These are the recommended techniques of performing string formatting in Python.

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- ~~Strings and Lists~~
- ~~String Formatting~~
- Control Flow
- Loops
- Functions



Control Flow

if and else

```
if expr:
  # Do something
elif other_expr:
  # Do another thing
else:
  # Do yet another thing
```

Truthiness and Falsiness

- Objects in Python are inherently "truthy" or "falsy" depending on their value; they can therefore be used in boolean expressions, and will be reduced to their truthiness or falsiness in the process.
  - "Zero" or "empty" (terms used loosely) objects are "falsy"; others are "truthy".

try and except

```
try:
  # Something dangerous
except ValueError:
  # Handle safely
```

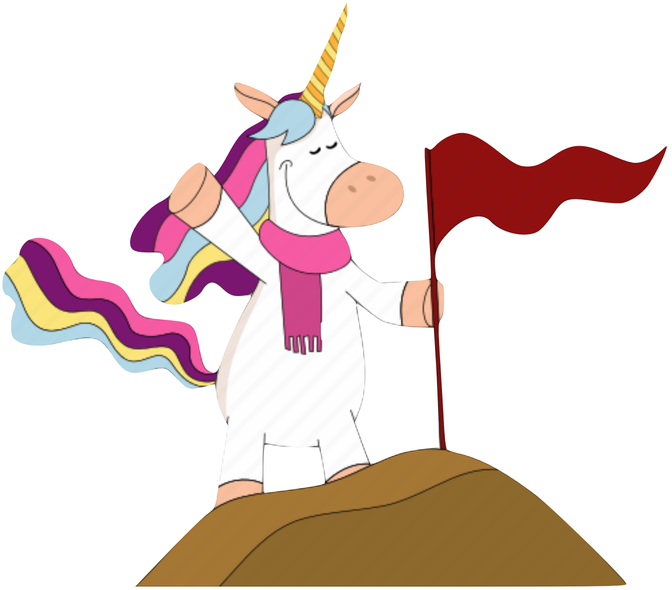
Control Flow: Summary

- Python provides control flow through two paradigms: `if/elif/else` and `try/except/finally`. A colon follows each control flow expression.
- In Python, it is typically easier to ask for forgiveness than for permission; therefore, using `try/except/finally` is often stylistically superior to error checking in advance with an `if/elif/else` block.
- Python does not use brackets or braces to track which code is inside a control flow statement: it parses the indentation to infer which code is inside which expressions.
  - Indented blocks in Python are denoted through four spaces (or one tab).

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- ~~Strings and Lists~~
- ~~String Formatting~~
- ~~Control Flow~~
- Loops
- Functions



Loops

`for` loops

```
for i in range(10):
  # i = 0, 1, ... 9
```

```
my_list = ["Parth", 5.0, 2]
for elem in my_list:
  # elem = "Parth", ... 2
```

`while` loops

```
while condition:
    # Do stuff
```

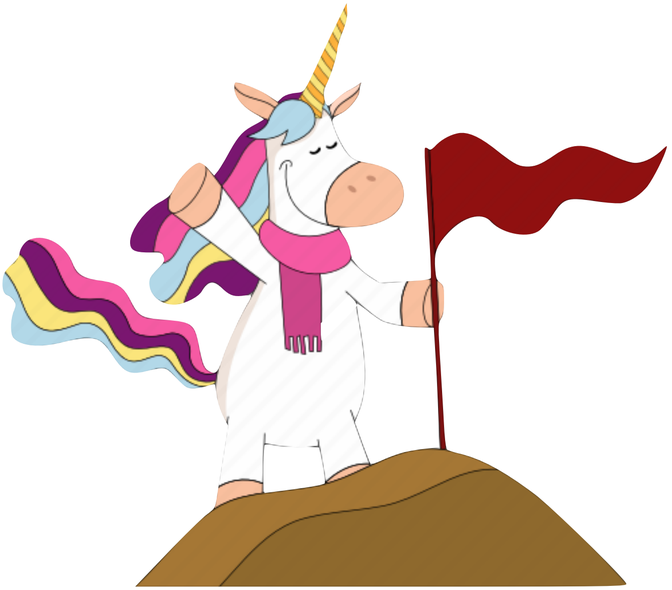
Loops: Summary

- Python supports both `for` and `while` loops.
  - `for` loops in Python do not, by default, use a loop counter: instead the loop variable takes on the value of sequential elements in an iterable.
- The `range` function generates an utterable over a range of numbers, using the syntax `range(start, stop[, step])`.
- The `break` statement breaks out of the smallest enclosing `for` or `while` loop; the `continue` statement skips to the top of the next iteration of the smallest enclosing loop.

Python Crash Course

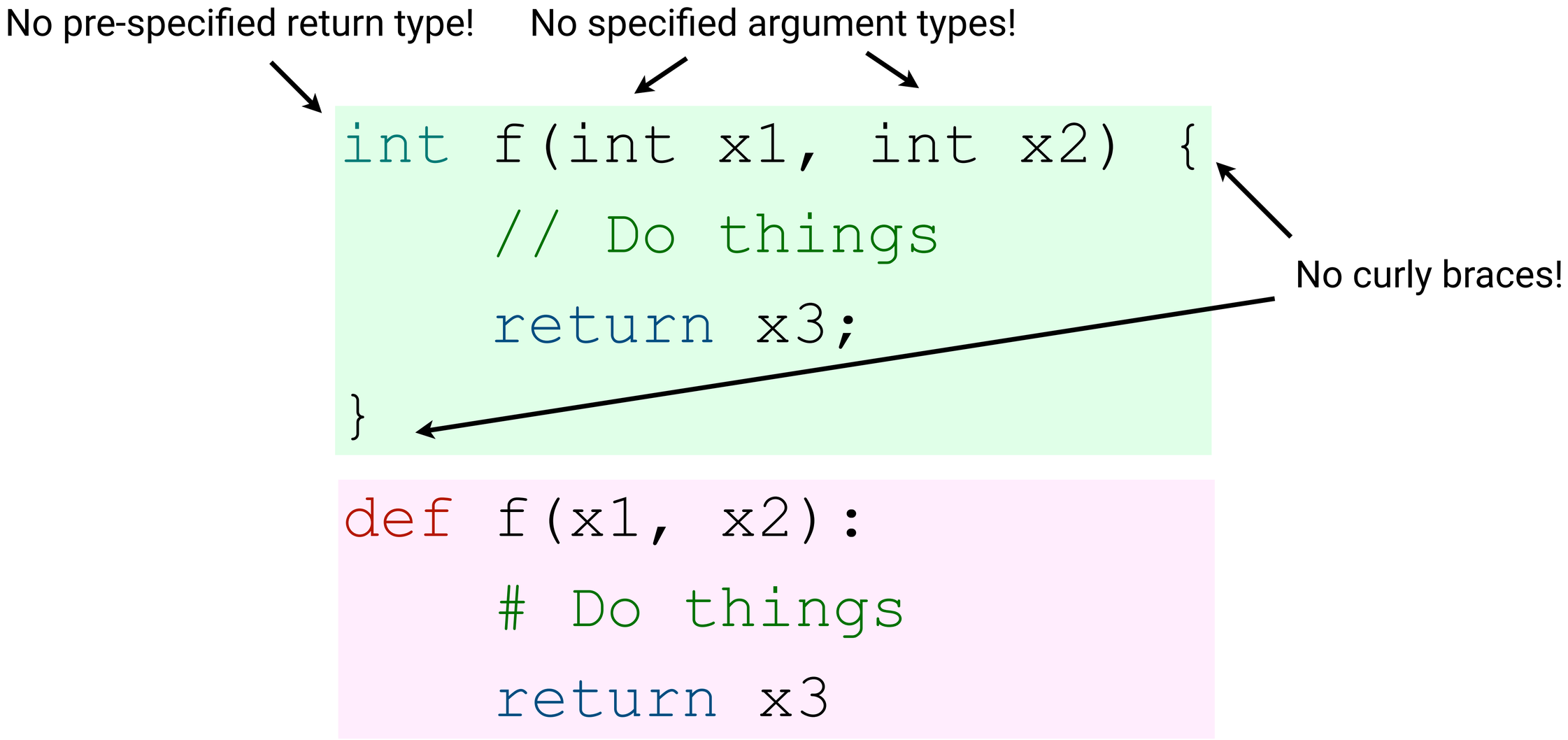
Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- ~~Strings and Lists~~
- ~~String Formatting~~
- ~~Control Flow~~
- ~~Loops~~
- Functions



Functions

Functions: Overview



Functions: Summary

- In Python, a function is defined using the `def` keyword. Code within a function is indented in the same manner as in a `for` or `while` loop.
- Function parameters do not have explicit types.
- A return type specification is not needed: one Python function can return values of different type depending on one or more conditions.
  - The `return` statement is optional: if it is not specified, the function will return `None`.

Python Crash Course

Python Crash Course

- ~~Interactive Interpreter~~
- ~~Variables and Types~~
- ~~Numbers and Booleans~~
- ~~Strings and Lists~~
- ~~String Formatting~~
- ~~Control Flow~~
- ~~Loops~~
- ~~Functions~~