Expressions

Chris Gregg

Based on slides by Chris Piech and Mehran Sahami

CS106A, Stanford University

Recall, add2numbers.py Program

```
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print(f"The total is {total}.")
```

Recall, add2numbers.py Program

```python
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))

    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print(f"The total is {total}.")
```

Recall, add2numbers.py Program

```
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))

    num2 = int(input("Enter second number: "))

    total = num1 + num2
    print(f"The total is {total}.")
```

Recall, add2numbers.py Program

```python
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    total = num1 + num2
    print(f"The total is {total}.")
```

- Often, this is how you'll see code that gets input
- But, what if I want to do more than add?
- It's time for the world of *expressions*

Today's Goal

1. Understanding arithmetic expressions
2. Using constants
3. Random number generation

Arithmetic Operators

```
num1 = 5
num2 = 2
```

- Operations on numerical types (**`int`** and **`float`**)
- Operators

			num3
`+`	"addition"	Ex.: `num3 = num1 + num2`	7
`-`	"subtraction"	Ex.: `num3 = num1 - num2`	3
`*`	"multiplication"	Ex.: `num3 = num1 * num2`	10
`/`	"division"	Ex.: `num3 = num1 / num2`	2.5
`//`	"integer division"	Ex.: `num3 = num1 // num2`	2
`%`	"remainder"	Ex.: `num3 = num1 % num2`	1
`**`	"exponentiation"	Ex.: `num3 = num1 ** num2`	25
`-`	"negation" (unary)	Ex.: `num3 = -num1`	-5

Precedence

- Precedence of operator (in order)

`()`	"parentheses"	highest
`**`	"exponentiation"	
`–`	"negation" (unary)	
`*, /, //, %`		
`+, –`		lowest

- Operators in same precedence category are evaluated left to right
  – Similar to rules of evaluating expressions in algebra

Precedence Example

$$x = \underbrace{1 + \underbrace{\underbrace{3 * 5}_{15} / 2}_{7.5}}_{8.5}$$

$$x \; \boxed{8.5}$$

Implicit Type Conversion

```
num1 = 5
num2 = 2
num3 = 1.9
```

- Operations on two **`int`**s (except **`/`**) that would result in an integer value are of type **`int`**

```
num1 + 7   = 12        (int)
```

  - Dividing (**`/`**) two **`int`**s results in a **`float`**, even if result is a round number (Ex.: **`6 / 2`** = **`3.0`**)

- If either (or both) of operands are **`float`**, the result is a **`float`**

```
num3 + 1   = 2.9      (float)
```

- Exponentiation depends on the result:

```
num2 ** 3   = 8     (int)
2 ** -1     = 0.5   (float)
```

Explicit Type Conversion

```
num1 = 5
num2 = 2
num3 = 1.9
```

- Use **float(*value*)** to create new real-valued number

```
float(num1)             = 5.0   (float)
```

  - Note that **num1** is not changed. We created a new value.

```
num1 + float(num2)      = 7.0   (float)
num1 + num2             = 7     (int)
```

- Use **int(*value*)** to create a new integer-valued number (<u>truncating</u> anything after decimal)

```
int(num3)               = 1     (int)
int(-2.7)               = -2    (int)
```

Float is Not Always Exact

```
num1 = 5
num2 = 2
num3 = 1.9
```

- What is type of: **`num3 - 1`**
  - Answer: **`float`**
- What is value of: **`num3 - 1`**
  - Answer: **`0.8999999999999999`**
  - WHAT?!

Expression Shorthands

```
num1 = 5
num2 = 2
num3 = 1.9
```

`num1 = num1 + 1`	same as	`num1 += 1`
`num2 = num2 - 4`	same as	`num2 -= 4`
`num3 = num3 * 2`	same as	`num3 *= 2`
`num1 = num1 / 2`	same as	`num1 /= 2`

- Generally:

  variable* = *variable operator (***expression***)

  is same as:

  variable operator**=** ***expression***

Let's consider an example

average2numbers.py

average2numbers.py

```python
"""
File: average2numbers.py
------------------------
This program asks the user for two numbers
and prints their average.
"""

def main():
    print("This program averages two numbers.")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    total = (num1 + num2) / 2
    print(f"The average is {total}.")

# This provided line is required at the end of a
# Python file to call the main() function.
if __name__ == '__main__':
    main()
```

Constants

```
INCHES_IN_FOOT = 12
PI = 3.1415
```

- Constants make code easier to read (good style):

  ```
  area = PI * (radius ** 2)
  ```

  - Written in all capital SNAKE_CASE with descriptive names
  - Constant are really variables that represent quantities that don’t change while the program is running
  - Can be changed between runs (as necessary)
    - "Hey, we need to compute a trajectory to get us to Mars"

    ```
    PI = 3.141592653589793
    ```

  - Code should be written with constants in a general way so that it still works when constants are changed

Example of Using Constants

```python
"""
File: constants.py
------------------
An example program with constants
"""


INCHES_IN_FOOT = 12

def main():
    feet = float(input("Enter number of feet: "))
    inches = feet * INCHES_IN_FOOT
    print(f"That is {inches} inches")

# This provided line is required at the end of a Python file
# to call the main() function.
if __name__ == '__main__':
    main()
```

Python `math` Library

```
import math
```

- math library has many built-in constants:

`math.pi`	mathematical constant π
`math.e`	mathematical constant *e*

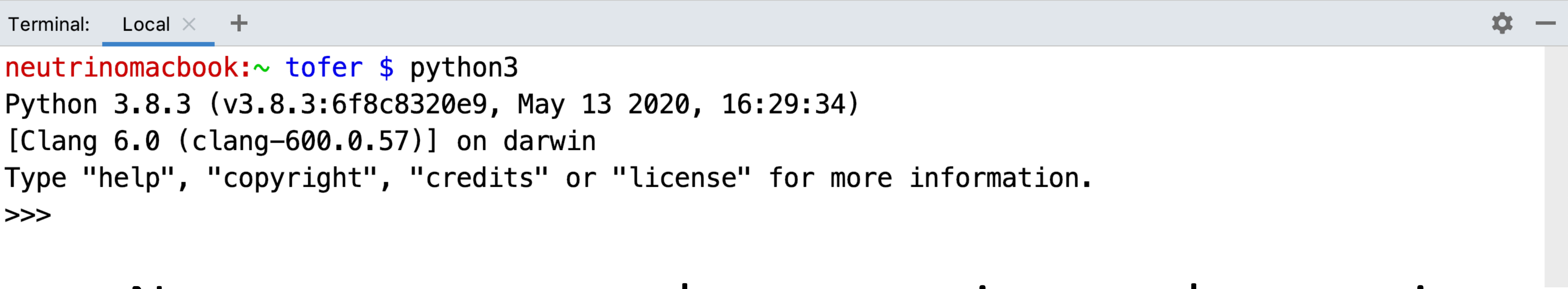
- and useful functions:

`math.sqrt(`x`)`	returns square root of x
`math.exp(`x`)`	returns e^x
`math.log(`x`)`	returns natural log (base *e*) of x

- These are just a few examples of what's in math
  - We can use the Python REPL to find out all the functions (see next slide)

The Python REPL

- The Python Read Evaluate Print Loop (REPL) is an easy way to quickly test things in Python, and it enables you to find out what functions exist in libraries (and get help on them)
- In the terminal, simply type `python3`:

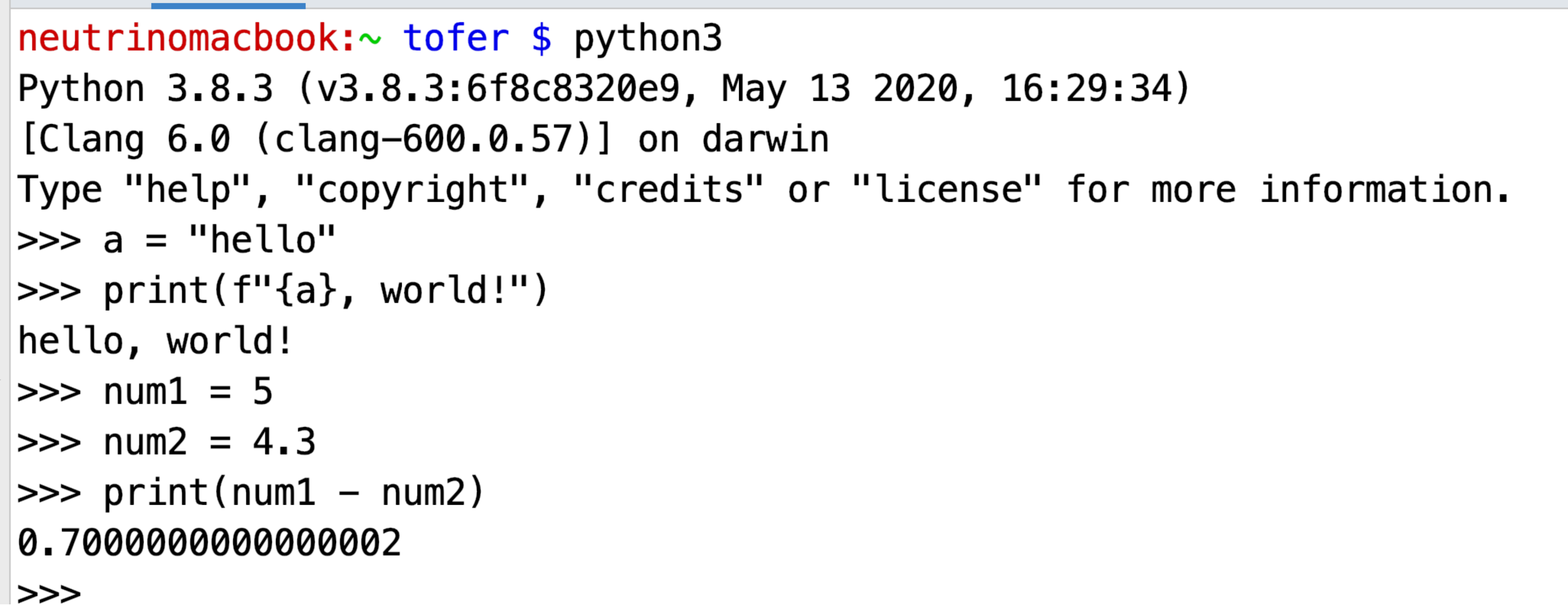


```
neutrinomacbook:~ tofer $ python3
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Now, you can type python expressions, and even write some code (but it is always much better to write programs in PyCharm itself)

The Python REPL

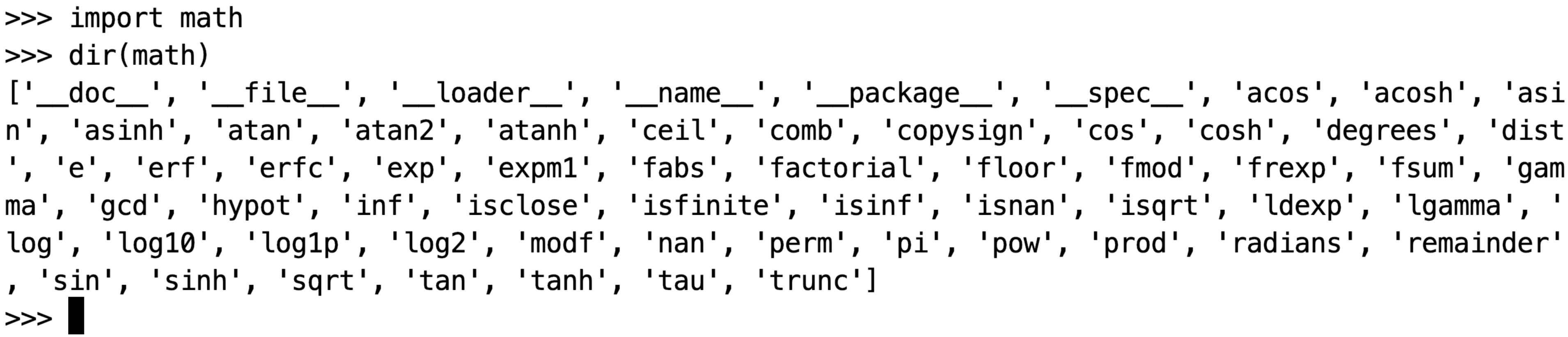
- REPL example:



```
neutrinomacbook:~ tofer $ python3
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print(f"{a}, world!")
hello, world!
>>> num1 = 5
>>> num2 = 4.3
>>> print(num1 - num2)
0.7000000000000002
>>>
```

The Python REPL

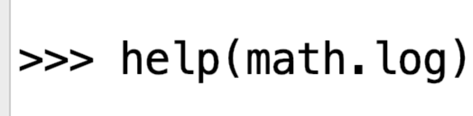
- If you import a library, you can use `dir(library_name)` to find out all the functions and constants the library has:



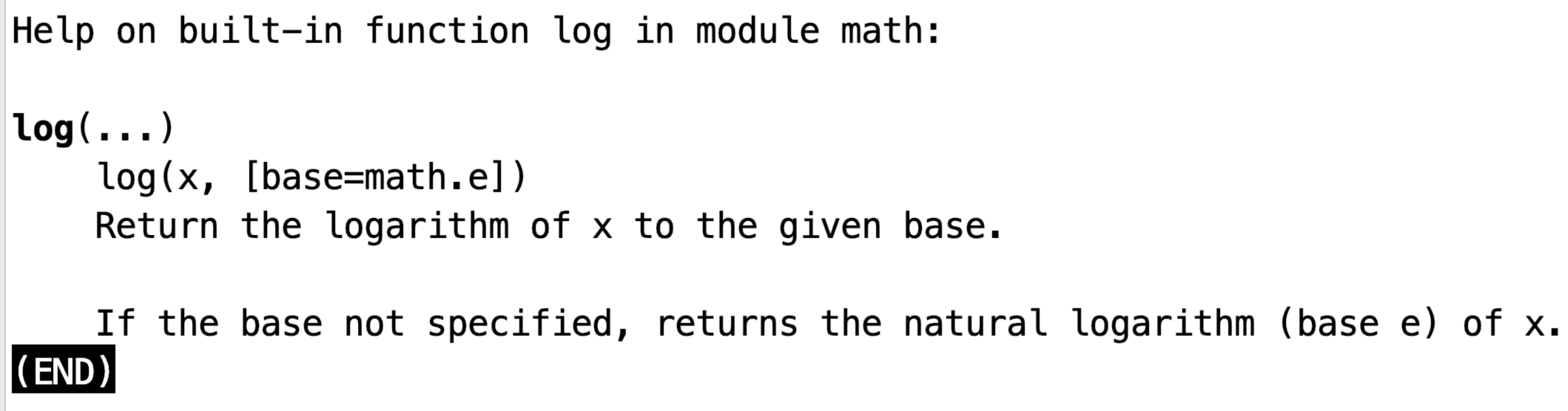
```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asi
n', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist
', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gam
ma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', '
log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder'
, 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> 
```

The Python REPL

- If you want help on a particular function, type `help(library_name.function_name)`



```
>>> help(math.log)
```



```
Help on built-in function log in module math:

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.
(END)
```

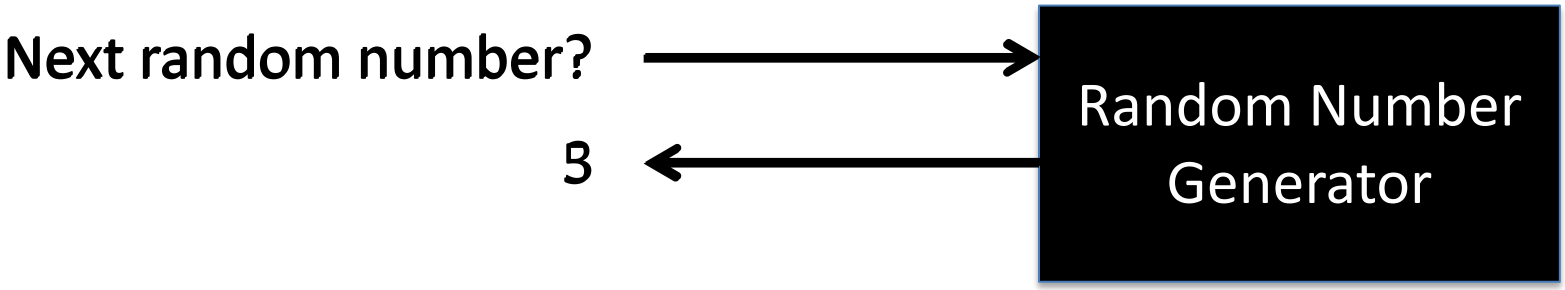
- Type the q key to get out of the help window

Example of Using `math` Library

```python
"""
File: squareroot.py
-------------------
This program computes square roots
"""


import math

def main():
    num = float(input("Enter number: "))
    root = math.sqrt(num)
    print(f"Square root of {num} is {root}")

# This provided line is required at the end of a Python file
# to call the main() function.
if __name__ == '__main__':
    main()
```

Random Number Generation

- Want a way to generate random number
  - Say, for games or other applications
- No "true" randomness in computer, so we have *pseudorandom* numbers
  - "That looks pretty random to me"
- Want "black box" that we can ask for random numbers



Next random number?

3

Random Number Generator

- Can "seed" the random number generator to always produce the same sequence of "random" numbers

Python random Library

```
import random
```

Function	What it does
random.randint(*min*, *max*)	Returns a random integer between *min* and *max*, inclusive.
random.random()	Returns a random real number (float) between 0 and 1.
random.uniform(*min*, *max*)	Returns a random real number (float) between *min* and *max*.
random.seed(*x*)	Sets "seed" of random number generator to *x*.

Let's consider an example

rolldice.py

Example of Using `random` Library

```python
"""
File: rolldice.py
-------------------
Simulate rolling two dice
"""

import random

NUM_SIDES = 6

def main():
    # setting seed is useful for debugging
    # random.seed(1)
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Dice have {NUM_SIDES} sides each.")
    print(f"First die: {die1}")
    print(f"Second die: {die2}")
    print(f"Total of two dice: {total}")
```

Today's Goal

1. Understanding arithmetic expressions
2. Using constants
3. Random number generation

Putting it all together: dicesimulator.py

What's Going On?

```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

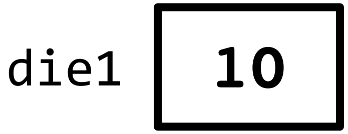
What's Going On?

```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

die1 10

What's Going On?

```python
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```



```
die1 in main() starts as: 10
```

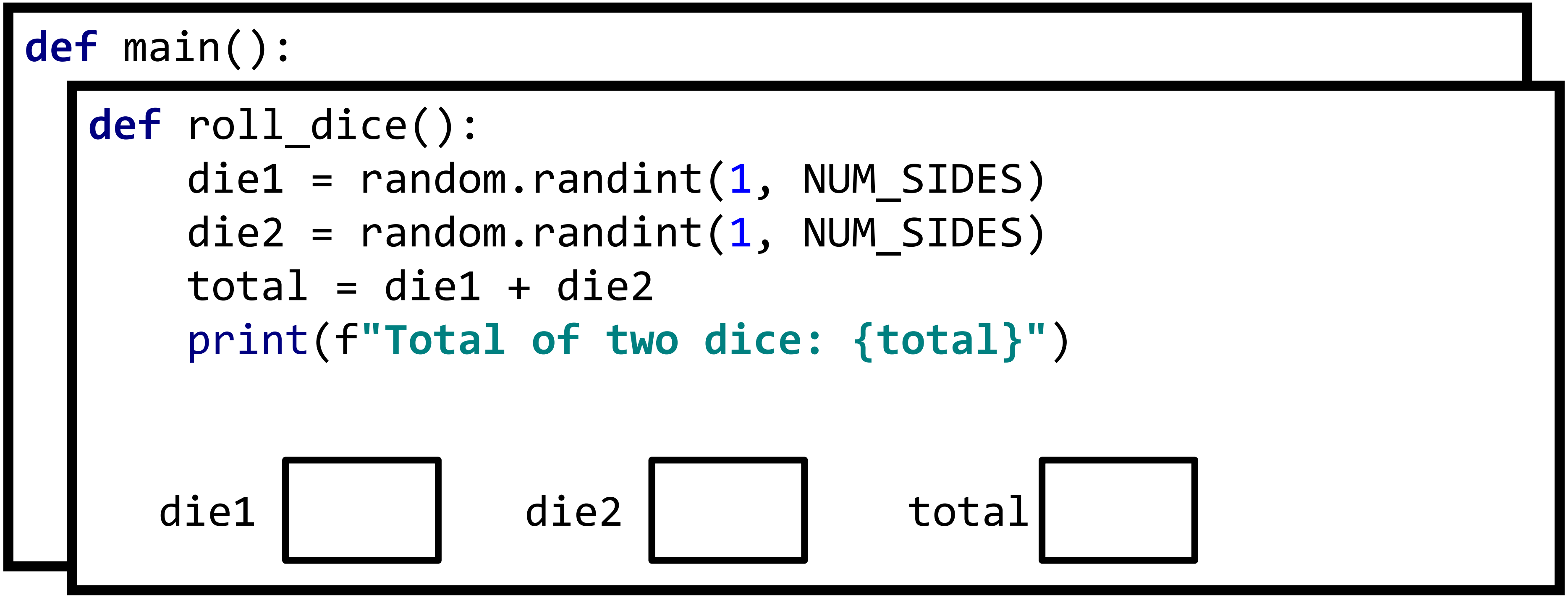
What's Going On?

```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

die1 10

```
die1 in main() starts as: 10
```

What's Going On?

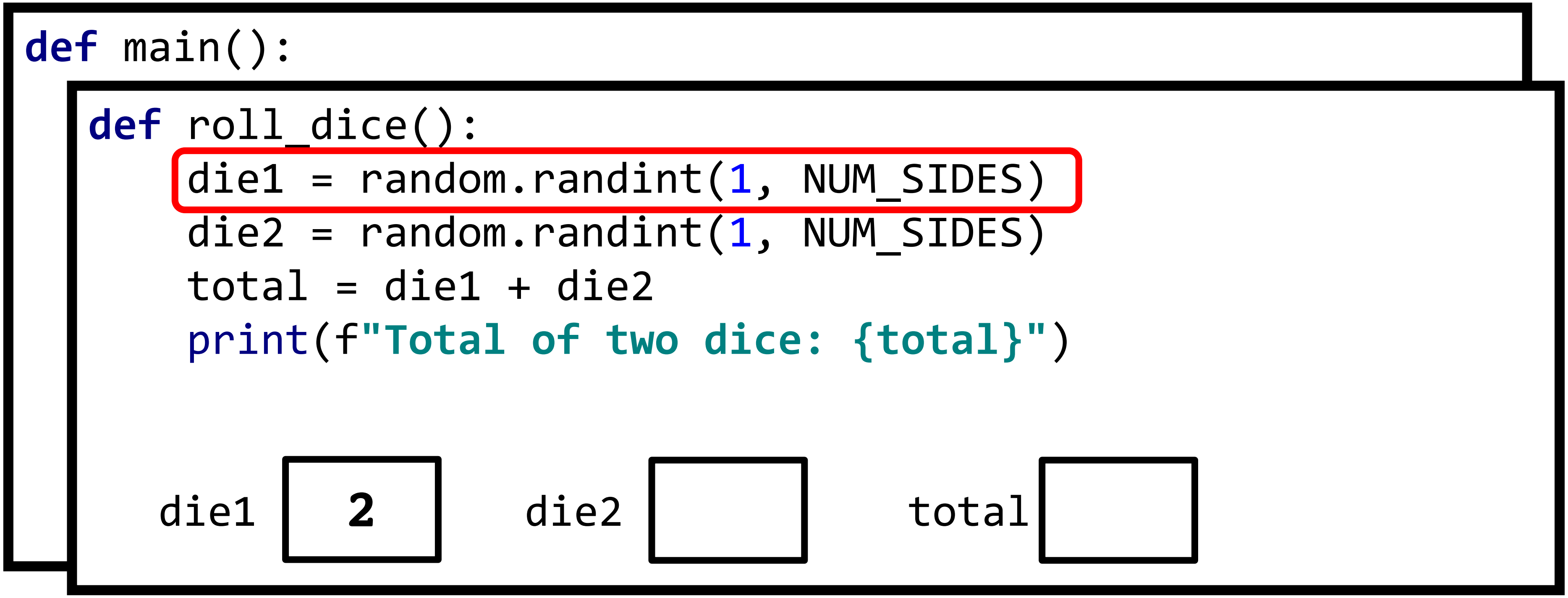


```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
```

What's Going On?



```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
```

What's Going On?

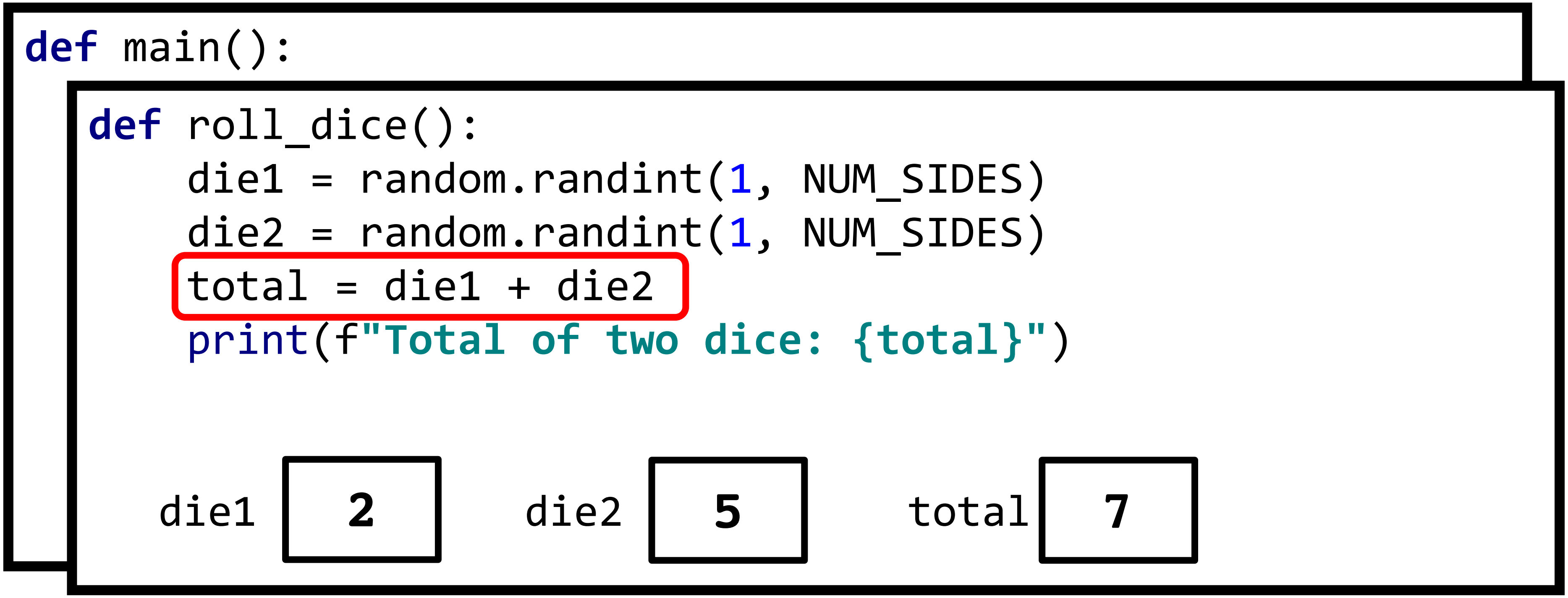
```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

die1 2 die2 5 total

```
die1 in main() starts as: 10
```

What's Going On?



```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
```

What's Going On?

```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1  2      die2  5      total  7
```

```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?

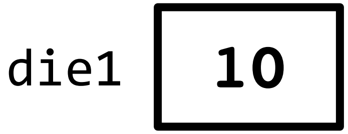
```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"Total of two dice: {total}")
```

die1 10

```
die1 in main() starts as: 10
Total of two dice: 7
```

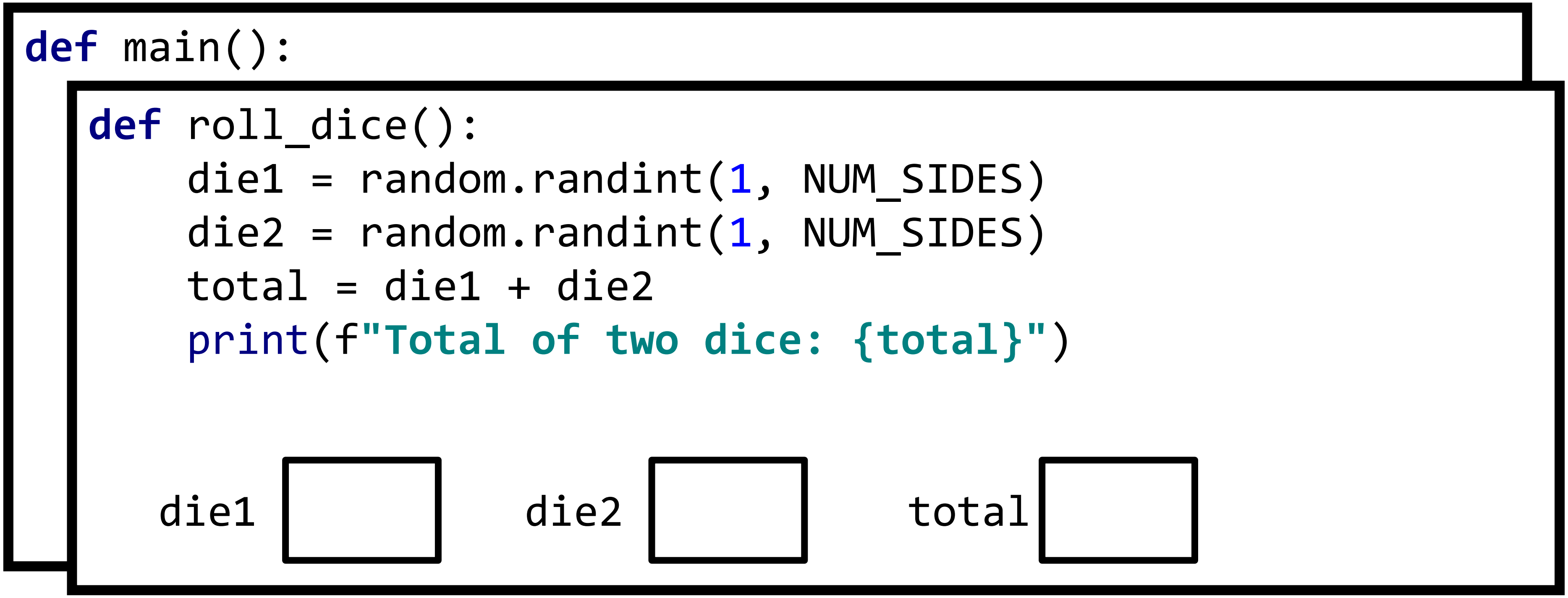
What's Going On?

```python
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"Total of two dice: {total}")
```



```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?



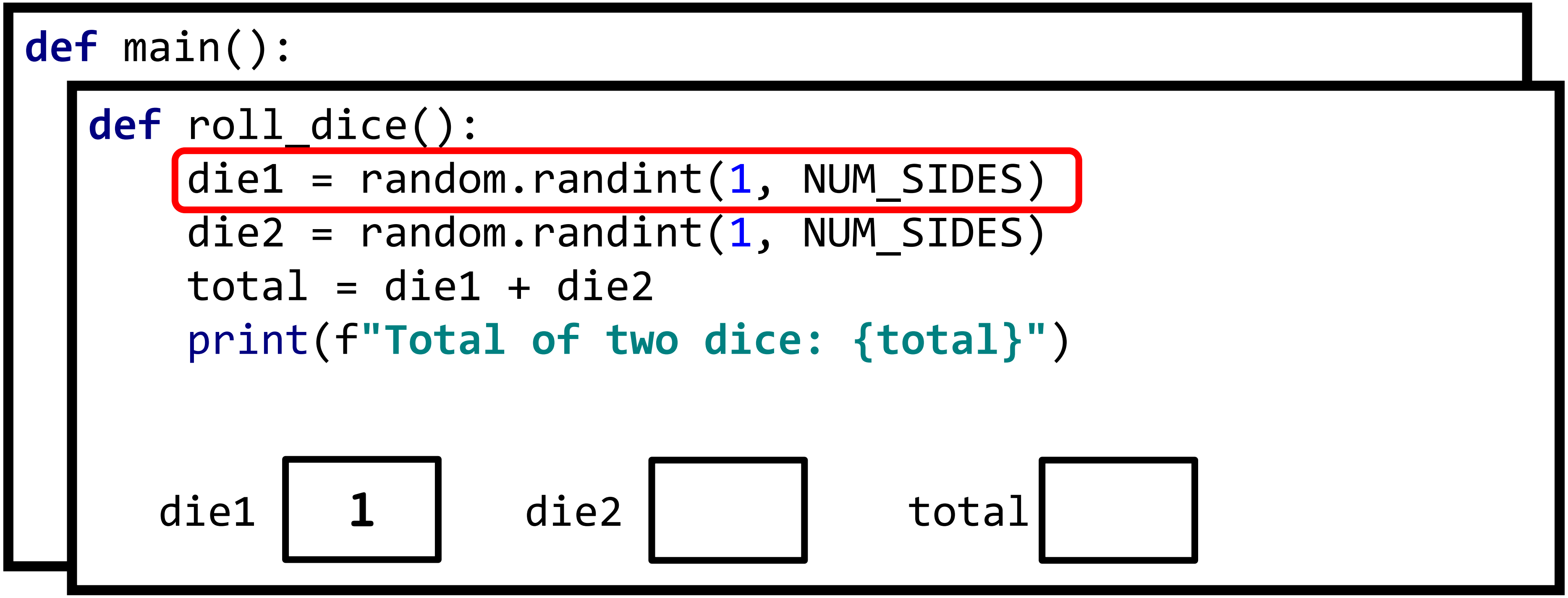
```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

die1 die2 total

```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?

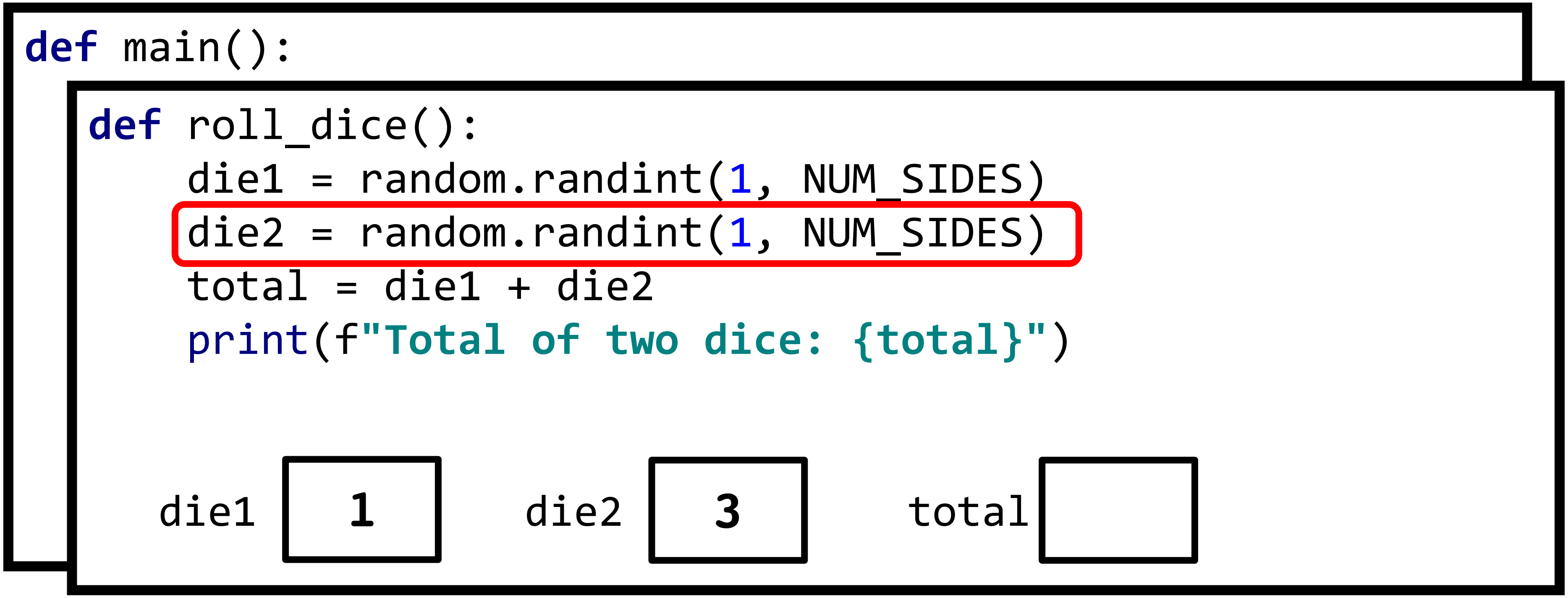


```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?



```
def main():
```

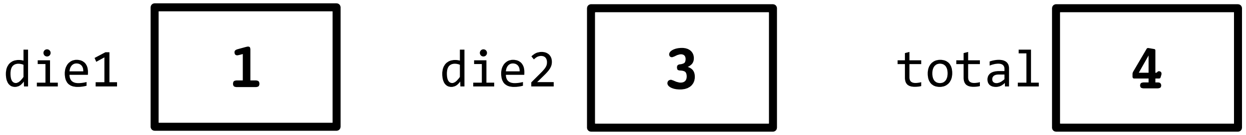
```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?

```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```



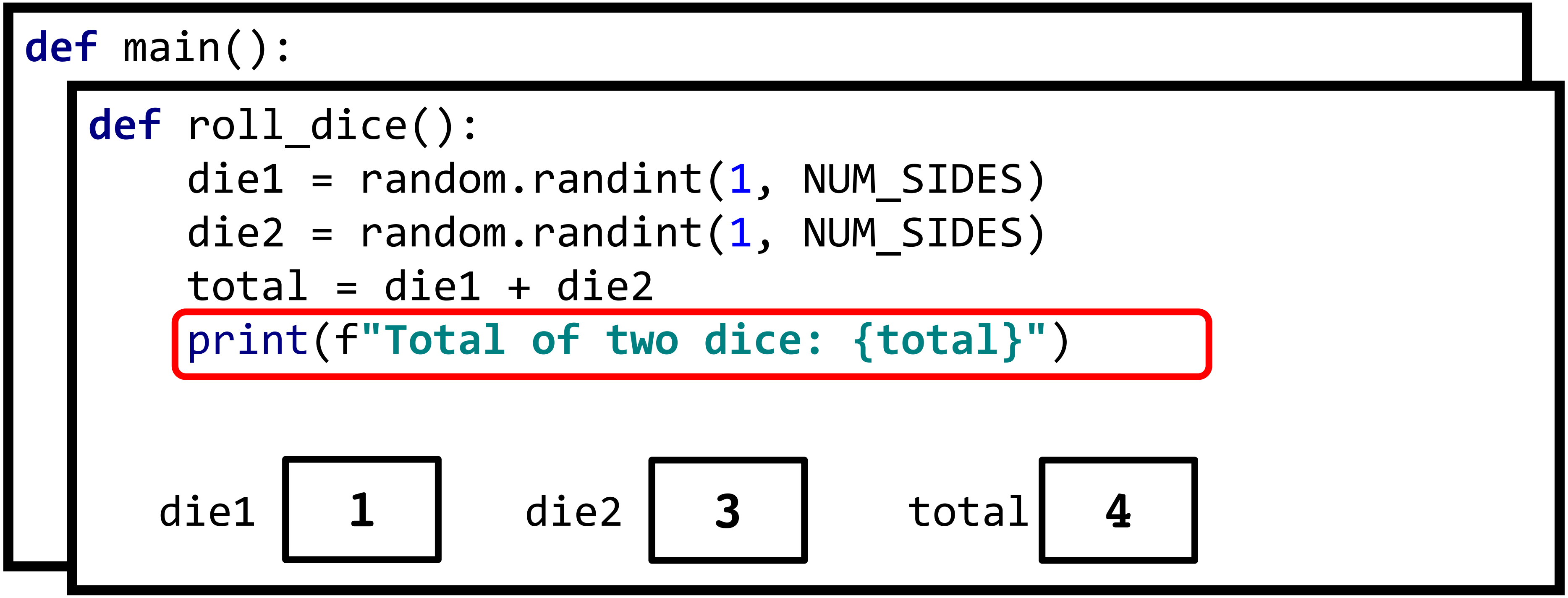
die1 1

die2 3

total 4

```
die1 in main() starts as: 10
Total of two dice: 7
```

What's Going On?



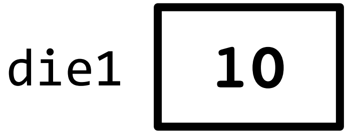
```
def main():
```

```
def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}")
```

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
```

What's Going On?

```python
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```



die1 10

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
```

What's Going On?

```python
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

die1 10

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
```

What's Going On?

```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

die1 10

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
Total of two dice: 5
```

What's Going On?

```
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")
```

die1 10

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
Total of two dice: 5
die1 in main() is: 10
```

You're rockin' it!