Control

9 / 27 / 21

Print and None

Demo

None Indicates that Nothing is Returned

- The special value **None** represents nothing in Python
- A function that does not explicitly return a value will return **None**
- *Careful:* **None** is *not displayed* by the interpreter as the value of an expression

```
>>> def does_not_return_square(x):
...     x * x
...
>>> does_not_return_square(4)
>>> sixteen = does_not_return_square(4)
>>> sixteen + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

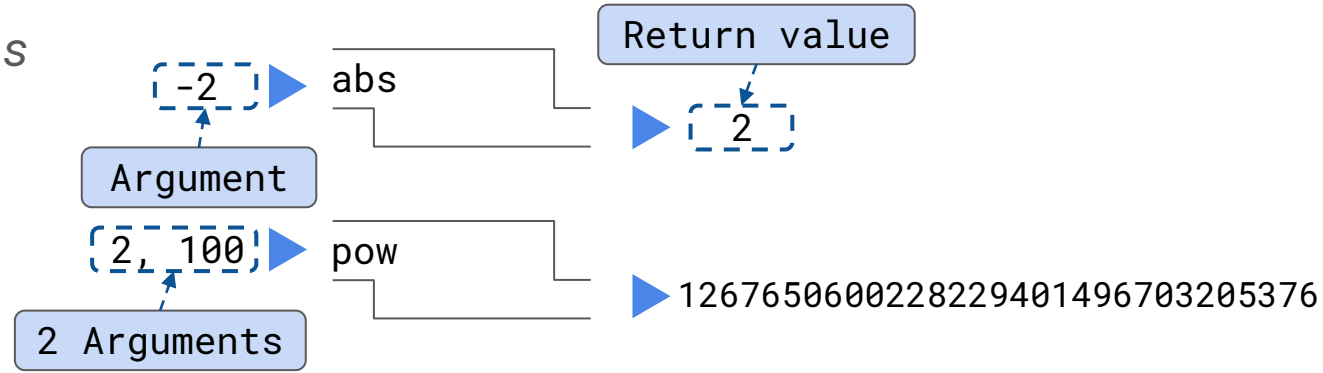
No return

None value is not displayed

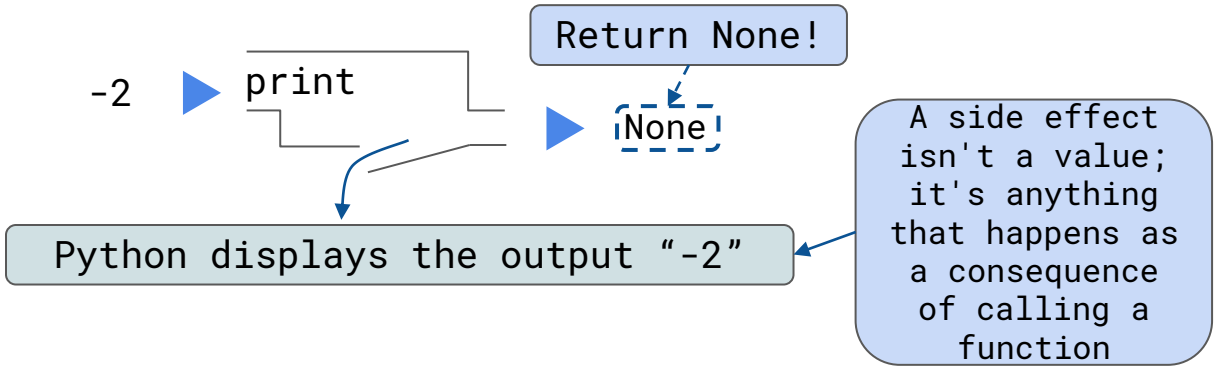
The name sixteen is now bound to the value None

Pure Functions & Non-Pure Functions

Pure Functions
just return values

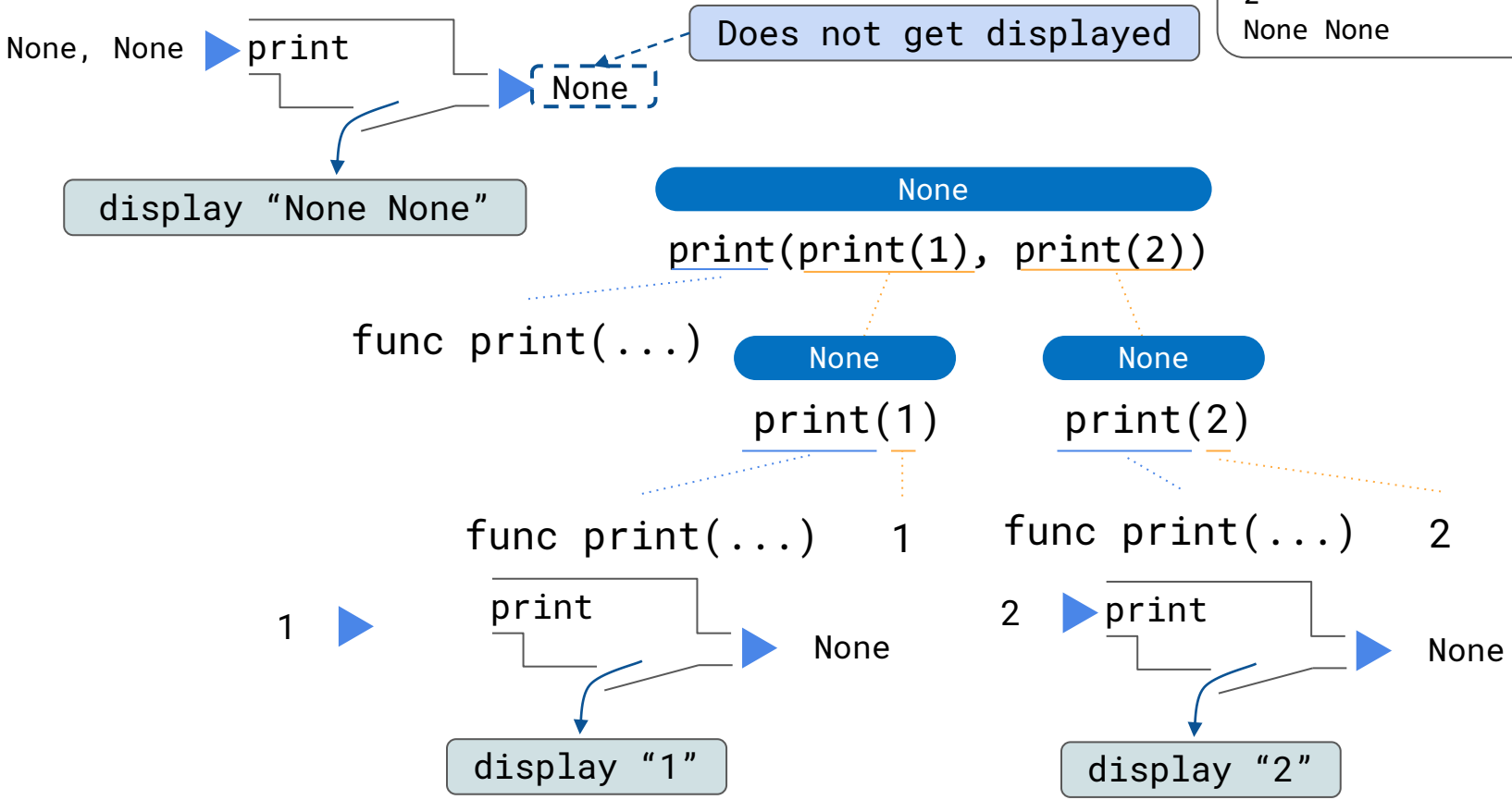


Non-Pure Functions
have side effects



Nested Expressions with Print

```
>>> print(print(1), print(2))
1
2
None None
```



print vs return

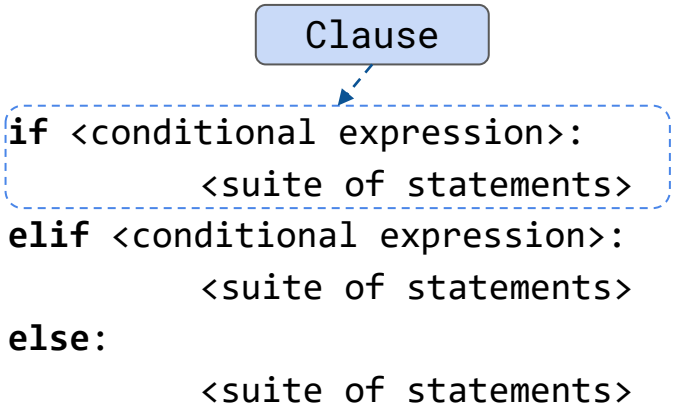
Demo

Control

Control

- **Expressions** in programs evaluate to values
- **Statements** are executed to perform actions
  - Ex: assignment and def statements
- With what we have seen so far, a lot of useful programs have been left out
- For example: returning ‘hot’, ‘warm’, or ‘cold’ depending on an argument temp
- To do this we introduce the concept of **control**
  - Special expressions and statements can **control** how the program is executed by the interpreter

Conditional statements (**if** statements)



Syntax:

- Always start with **if** clause
- Zero or more **elif** clauses
- Zero or one **else** clause, always at the end

Execution Rule for Conditional Statements:

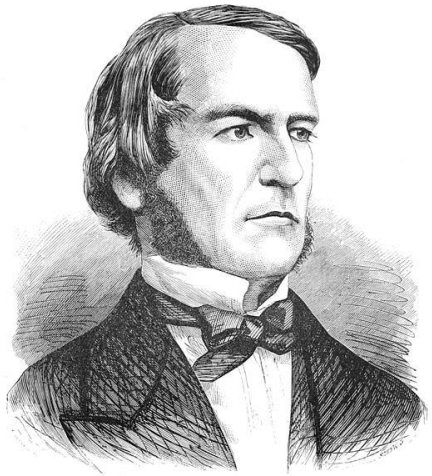
Each header is considered in order

1. Evaluate the header's conditional expression if the header is not an **else**
2. If the expression evaluates to true or the header is an else, execute the suite and skip the remaining headers

`if` examples

Demo

Boolean Contexts



George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

Boolean context is any place where an expression is evaluated to check if it's a True value or a False value

False values in Python: False, None, 0, '' (more to come)
True values: everything else

Boolean Expressions

Boolean expressions contain special operators **and**, **or**, **not**

- <exp1> **and** <exp2> **and** <exp3> **and** ...
  - Evaluate to the first false value.
  - If none are false, evaluates to the last expression
- <exp1> **or** <exp2> **or** <exp3> **or** ...
  - Evaluate to first true value.
  - If none are true, evaluates to the last expression
- **not** <exp>
  - Evaluates to True if <exp> if a *false value* and False if <exp> is a *true value*

Short-Circuiting

Demo

Iteration

Demo

Demo

While statements

```
i, total = 0, 0
while i < 3:
        i = i + 1
        total = total + i
```

Execution Rule for While Statements:

1. Evaluate the header's expression
2. If it is a true value, execute the (whole) suite, then return to step 1

Python Tutor

The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

```
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1"""
    pred, curr = 0, 1  # 0th and 1st Fibonacci numbers
    k = 1              # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The next Fibonacci number is the sum of the current one and its predecessor

Python Tutor

Summary

- print vs None
  - None represents when an expressions doesn't evaluate to a value
  - print displays values in the interpreter
- Control
  - Allow for the interpreter to selectively or repetitively execute parts of a program
- Iteration
  - A particular variant of control which is based in the idea of repeating operations to compute a value