Control

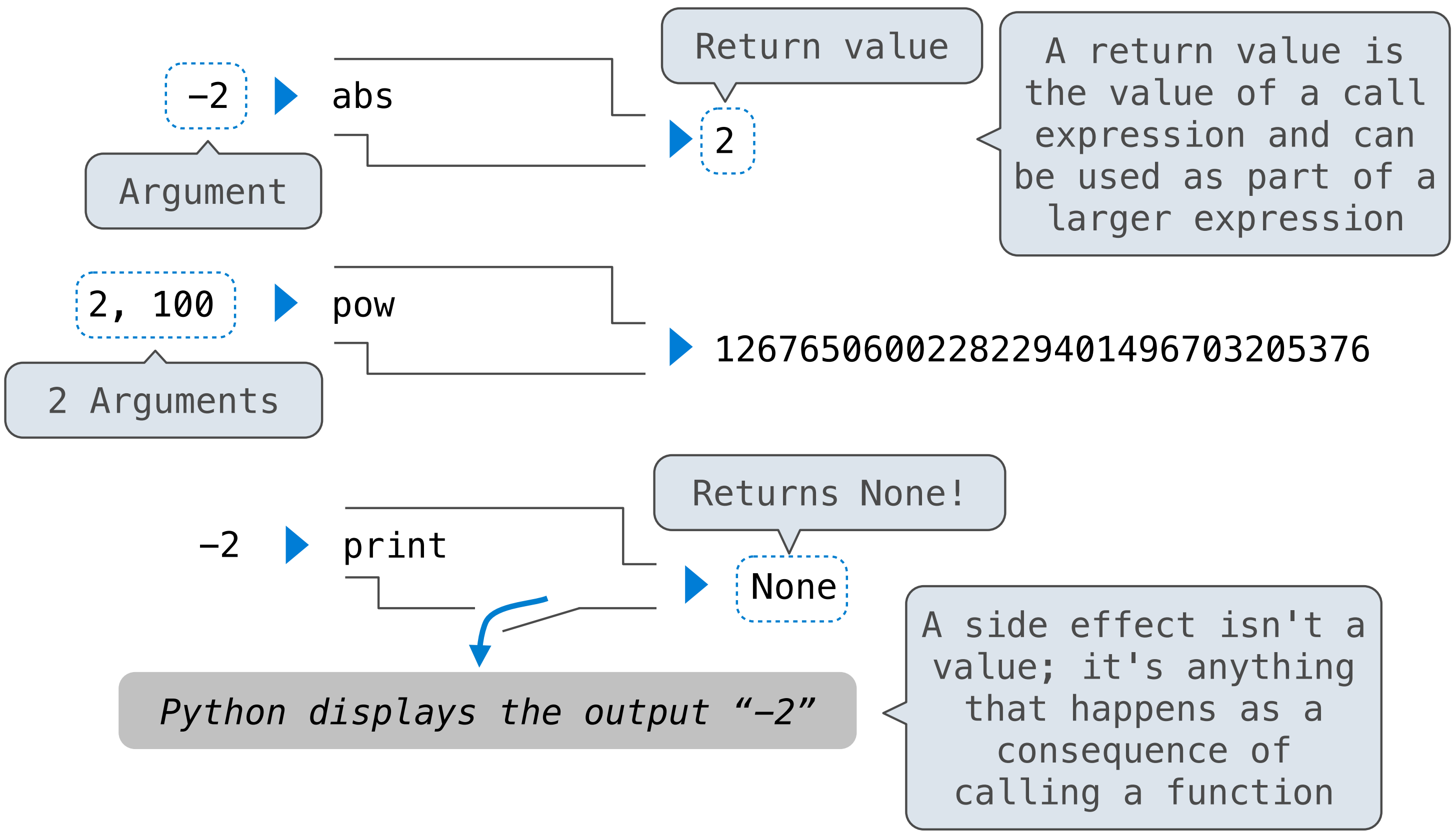
Announcements

Print and None

Pure Functions & Non-Pure Functions

Pure Functions
just return values

Non-Pure Functions
have side effects



Example: Print Then Return

Implement a function h(x) that first prints, then returns, the value of f(x).

```
def h(x):
    return print(f(x))
```

(A)

```
def h(x):
    print(f(x))
    return f(x)
```

(B)

```
def h(x):
    y = f(x)
    print(y)
    return y
```

(C)

What's a function f for which implementations (B) and (C) would have different behavior?

```
>>> h(2)
...
```

```
>>> h(2)
...
```

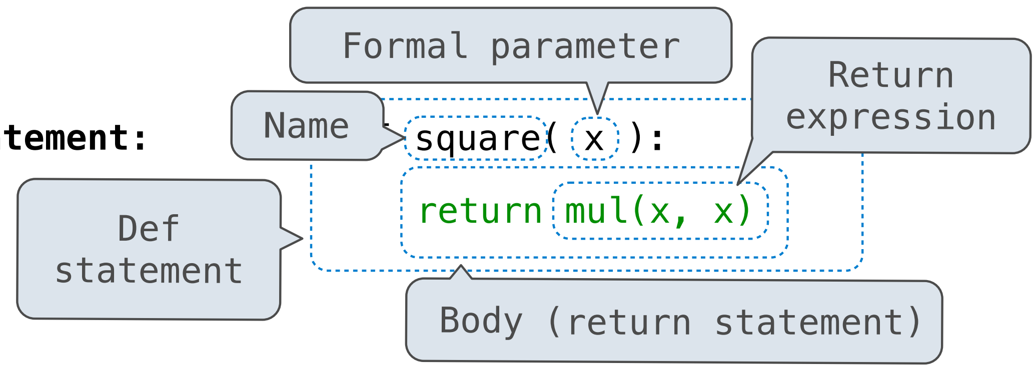
(Demo)

Multiple Environments

Life Cycle of a User-Defined Function

What happens?

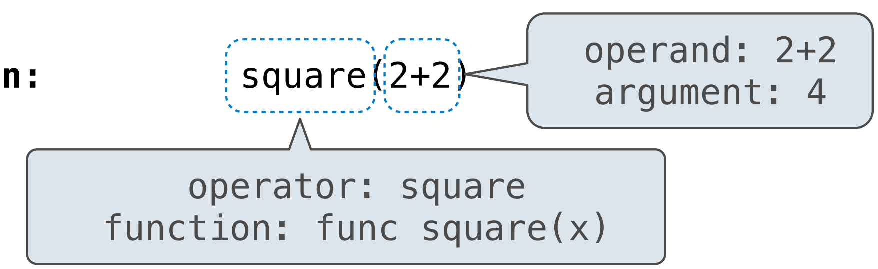
Def statement:



A new function is created!

Name bound to that function in the current frame

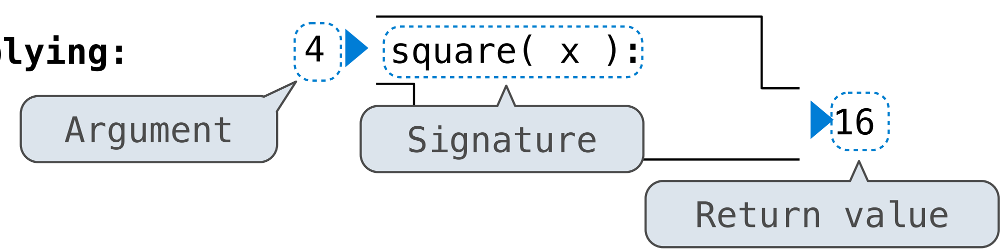
Call expression:



Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

Calling/Applying:



A new frame is created!

Parameters bound to arguments

Body is executed in that new environment

https://pythontutor.com/cp/composingprograms.html#code=def%20f%28x%29%3A%0A%20%20%20%20return%20square%28x%20%2B%20square%28y%20%2B%201%29%29%0A%20%20%20%20%0Adef%20square%28z%29%3A%0A%20%20%20%20y%20%3D%20z%20*%20z%0A%20%20%20%20return%20y%0A%20%20%20%20%0Ax,%20y,%20z%20%3D%201,%202,%203%0Aprint%28f%28z%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Control

Conditional Statements

Conditional statements (often called "If" Statements) contain statements that may or may not be evaluated.

		x=10	x=1	x=-1

`if x > 2:`
`    print('big')`
`if x > 0:`
`    print('positive')`
	Two separate (unrelated) conditional statements	big
		positive
			positive	

`if x > 2:`
`    print('big')`
`elif x > 0:`
`    print('positive')`
	One statement with two clauses: if and elif
	Only one body can ever be executed
		big	positive	

`if x > 2:`
`    print('big')`
`elif x > 0:`
`    print('positive')`
`else:`
`    print('not pos')`
	One statement with three clauses: if, elif, else
	Only one body can ever be executed
		big	positive	not pos

While Statements

While statements contain statements that are repeated as long as some condition is true.

Important considerations:

- How many separate names are needed and what do they mean?
- The while condition **must eventually become a false value** for the statement to end (unless there is a return statement inside the while body).
- Once the while condition is evaluated, the entire body is executed.

Names and their initial values

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

The while condition is evaluated before each iteration

A name that appears in the while condition is changing

Executed even when is set to 3

Example: Nice Numbers

Nice Numbers

Rounding off 2,799 to 2,800 makes it nice.

Definition: A nice number doesn't have 98 or 99 or 01 or 02 among its digits.

Not-so-nice numbers: **99** **2,799** **5,016** **9,902** **1,200,456** **98,402,001**

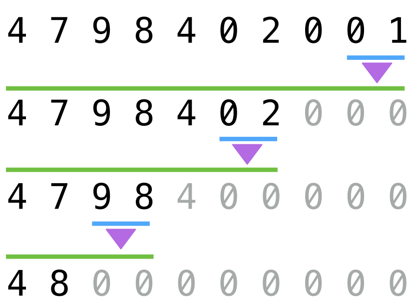
Nicer versions: **100** **2,800** **5,000** **10,000** **1,200,000** **100,000,000**

These numbers are nice enough already and unaffected: **755** **2,859** **45,622,895**

Implement **nice**, which takes a positive integer n. It returns the nearest nice number to n.

- For numbers that end in 98 or 99 or 01 or 02, round to the nearest 100.
- Look for 98 or 99 or 01 or 02 among the digits that aren't at the end.

To solve a problem, describe a process and work through an example:



(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20nice_end%28n%29%3A%0A%20%20%20%20%22Round%20n%20ending%20in%2098,%2099,%2001,%20or%2002%20to%20the%20nearest%20100.%22%0A%20%20%20%20if%20n%20%25%20100%20%3E%3D%2098%3A%0A%20%20%20%20%20%20%20%20return%20n%20//%20100%20*%20100%20%2B%20100%0A%20%20%20%20elif%20n%20%25%20100%20%3C%3D%202%3A%0A%20%20%20%20%20%20%20%20return%20n%20//%20100%20*%20100%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20n%0A%0Adef%20nice%28n%29%3A%0A%20%20%20%20%22Find%20the%20closest%20nice%20number%20to%20n.%22%0A%20%20%20%20rest%20%3D%20n%0A%20%20%20%20k%20%3D%200%0A%20%20%20%20while%20rest%20%3E%2010%3A%0A%20%20%20%20%20%20%20%20if%20nice_end%28rest%29%20%25%20100%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20n%20%3D%20nice_end%28rest%29%20*%2010%20**%20k%0A%20%20%20%20%20%20%20%20rest,%20k%20%3D%20rest%20//%2010,%20k%20%2B%201%0A%20%20%20%20return%20n%0A%0Anice%284798402001%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Example: Prime Factorization

Prime Factorization

```
Each positive integer n has a set of prime factors: primes whose product is n

...
8  = 2 * 2 * 2
9  = 3 * 3
10 = 2 * 5
11 = 11
12 = 2 * 2 * 3
...

One approach: Find the smallest prime factor of n, then divide by it

858  = 2 * 429  = 2 * 3 * 143  = 2 * 3 * 11 * 13

(Demo)
```