Warmup (L2)

Print `"Hello, world!"`, with the space, *without* putting a space in your string.

That is,

```
print("Hello, world!")
```

But, no space allowed here!
Get it to print the space without writing this space.

Reusing (functions)

CS114 M1

Back to Pythagoras

- Note how I wrote the code:

```
sqrt(a**2 + b**2)
```

- What's this *a* and *b*?
- We want to use this code for any value of *a* and *b*
- Right now, if we need it twice, we have to rewrite this code!

Functions

- We already saw functions such as `print` and `sqrt`
- Now let’s write one of our own!

Functions

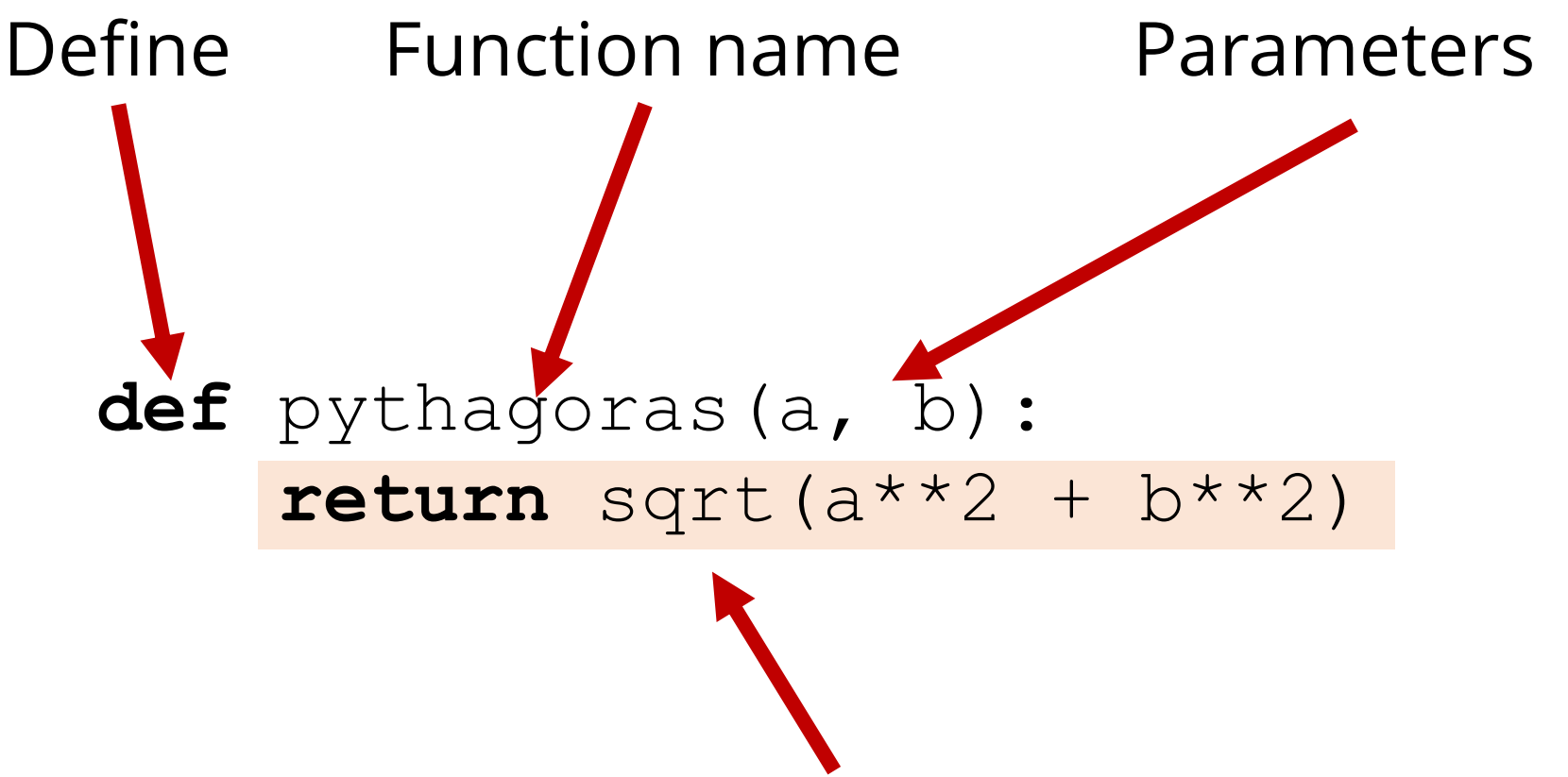
- We already saw functions such as `print` and `sqrt`
- Now let's write one of our own!

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
```

Functions

- We **def**ine a function with the keyword **`def`**
  - (A keyword is a word that has a special meaning in the programming language, other than just a name)
- We want this function to be *parameterized*
  - Similar to `sqrt`. We need to be able to tell `sqrt` what we want to square-root, and we need to be able to tell `pythagoras` the sides of our triangle

Functions


Define
Function name
Parameters
def pythagoras(a, b):
return sqrt(a**2 + b**2)
Body of the function (what it actually does)

Functions

Header

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
```

Body is *indented* past the header
(that's how Python knows it's the body!)

Functions

- Defining the function doesn't make anything happen
- We have to *call* the function for it to run

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
[…]
pythagoras(3, 4)
```

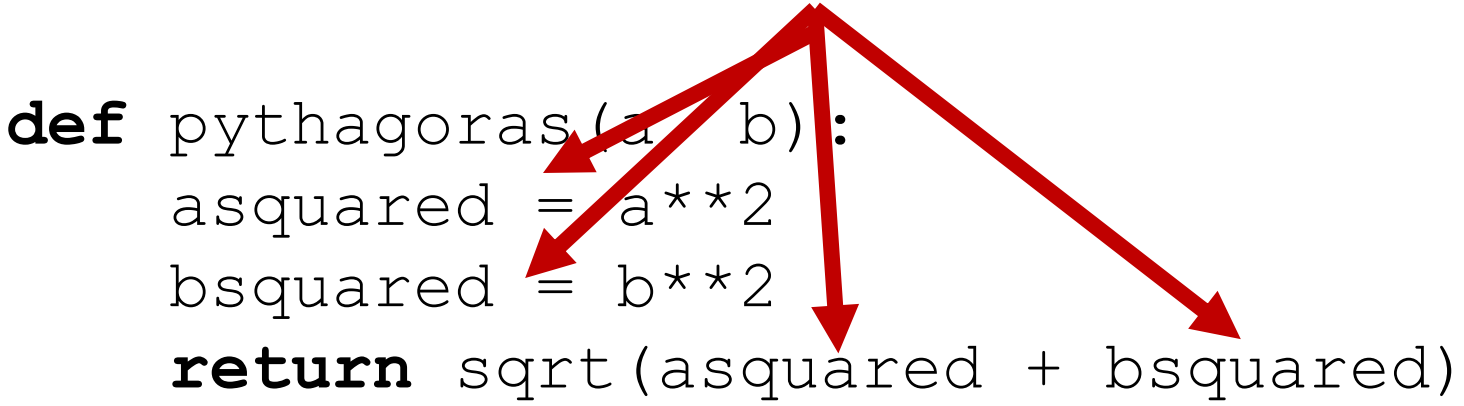
Functions

- A function can be any number of steps

```
def pythagoras(a, b):
    asquared = a**2
    bsquared = b**2
    return sqrt(asquared + bsquared)
```

Functions

Functions can define and use their own variables



```
def pythagoras(a, b):
    asquared = a**2
    bsquared = b**2
    return sqrt(asquared + bsquared)
```

Functions and variables

- They are the function's *own variables*
- They don't exist outside of the function:

```
def pythagoras(a, b):
    asquared = a ** 2
    bsquared = b ** 2
    return sqrt(asquared + bsquared)
pythagoras(3, 4)
print(asquared) # ERROR!
```

Functions and variables

- Parameters are also variables
- You can overwrite them

```
def pythagoras(a, b):
    a = a ** 2
    b = b ** 2
    return sqrt(a + b)
```

- This is usually confusing and should usually be avoided, but use common sense

Local vs. global

- Variables within a function are *local variables* of that function
- Variables outside the function (defined for the whole kernel) are *global variables*

Pedantry aside

- I've used the terms "parameter" and "argument"
  - In `pythagoras`, `a` and `b` are parameters
  - The value you actually pass in is the argument: the argument fills the parameter
- The confusion: if we talk about what `pythagoras` does, we'll talk about `a` and `b` as their future values, the arguments
- Often used interchangeably, but technically not the same

Functions calling functions

- Our `pythagoras` function calls `sqrt`
- We can also call our own functions in functions

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
[…]
def pythagoras3(a, b, c):
    return sqrt(pythagoras(a, b)**2 + c**2)
```

Returning

- Here, we use a call to `pythagoras` as a value in a calculation. What actual number is used in the calculation?

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
[…]
def pythagoras3(a, b, c):
    return sqrt(pythagoras(a, b)**2 + c**2)
```

Returning

- The **`return`** statement of the `pythagoras` function gives a value to whoever called `pythagoras`

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
[…]
def pythagoras3(a, b, c):
    return sqrt(pythagoras(a, b)**2 + c**2)
```

Returning

- ***WARNING!*** Returning *ends* the function, even if there are more statements left!

```
def pythagoras(a, b):
    return sqrt(a**2 + b**2)
    print("This will never be printed!")
```

- Why would you want this? We'll see when we get to decision-making.

Functions calling functions

- `print` is also a function
- Very useful for understanding and debugging code!

```
def pythagoras3(a, b, c):
    h = pythagoras(a, b)
    print("Hypotenuse of one face:", h)
    return sqrt(h**2 + c**2)
```

To return or not?

- A function doesn't have to explicitly return

```
def pythagoras(a, b):
    sqrt(a**2 + b**2)
```

- But, if it doesn't, the value it returns is "`None`", a special value that means "there is no value".

None is Hell

- Why have functions return `None`?
- We want functions that are just there to *do* something, rather than *returning* something…
- but Python has no way of distinguishing these two kinds of functions.
- So, Python needed *something* for `x = print("Whoops!")` to do.

None is Hell

- You can store `None` in a variable and Python won't report anything wrong
- You can pass `None` as an argument to a function, and it'll work until (and unless!) it's used
- Forgetting to **`return`** causes a problem to arise *distant from the actual error in the code*

Bugs, bugs, bugs!

CS114 M1

Debugging

- Time for debugging!

```
from math import sqrt
def pythagoras(a, b):
    sqrt(a**2 + b**2)
def pythagoras3(a, b, c):
    return sqrt(pythagoras(a, b)**2 + c**2)
print(pythagoras3(4, 5, 6))
```

Debugging

One thing to be careful of when debugging: errors happen when code *runs*, so if you don't run a buggy function, you won't see the problem!

```
from math import sqrt
def pythagoras(a, b):
    sqrt(a**2 + b**2)
def pythagoras3(a, b, c):
    return sqrt(pythagoras(a, b)**2 + c**2)
print(pythagoras(4, 5))
```

Avoiding bugs

- Develop incrementally (one small step at a time), using `print` to spot-check
- Let's make a `distance` function (distance between two points) incrementally…

```
def distance(x1, y1, x2, y2):
    …
```

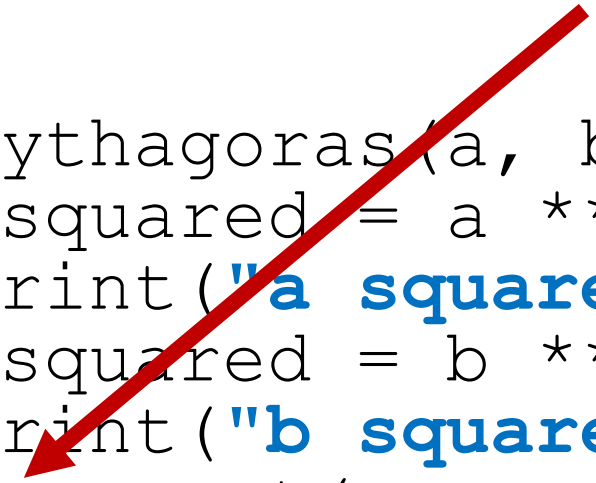
Avoiding bugs: printing

- Make sure whatever you print is descriptive/unique enough that you know which is which

```
def pythagoras(a, b):
    asquared = a ** 2
    print("a squared:", asquared)
    bsquared = b ** 2
    print("b squared:", bsquared)
    r = sqrt(asquared + bsquared)
    print("result:", r)
    return r
```

Avoiding bugs: printing

Don’t be afraid to introduce variables just so that you can print something from the middle of a calculation!



```
def pythagoras(a, b):
    asquared = a ** 2
    print("a squared:", asquared)
    bsquared = b ** 2
    print("b squared:", bsquared)
    r = sqrt(asquared + bsquared)
    print("result:", r)
    return r
```

Avoiding bugs: printing

- When you're done debugging, make sure you remove the prints. They'll confuse our tests!
- You can also *comment out* prints, so you can remove them without forgetting them

```
def hypotenuse(a, b):
    # print("a was", a)
    return sqrt(a**2 + b**2)
```

Avoiding bugs: documentation

- Part of avoiding bugs is *good documentation*
- You shouldn't name a function "`pythagoras`"
  - The Pythagorean theorem is an implementation detail
  - When you're calling the function, you don't care how it's implemented
  - It should've been named "`hypotenuse`"

Avoiding bugs: documentation

- Remember the `help` function?
- We can (and should!) document our functions in the same way, with *docstrings*
- Let's add a docstring to `pythagoras`

Avoiding bugs: documentation

```
def pythagoras(a, b):
    """
    Return the length of the
    hypotenuse of a right
    triangle with side lengths a
    and b.
    """
    return sqrt(a**2 + b**2)
```