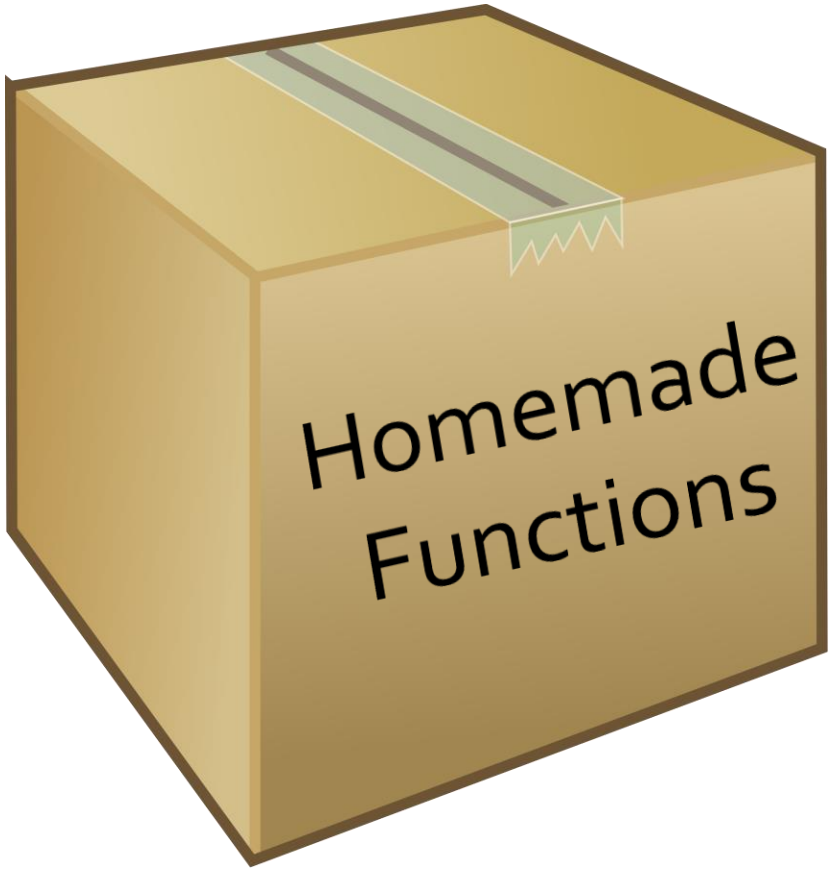
DEFINING FUNCTIONS

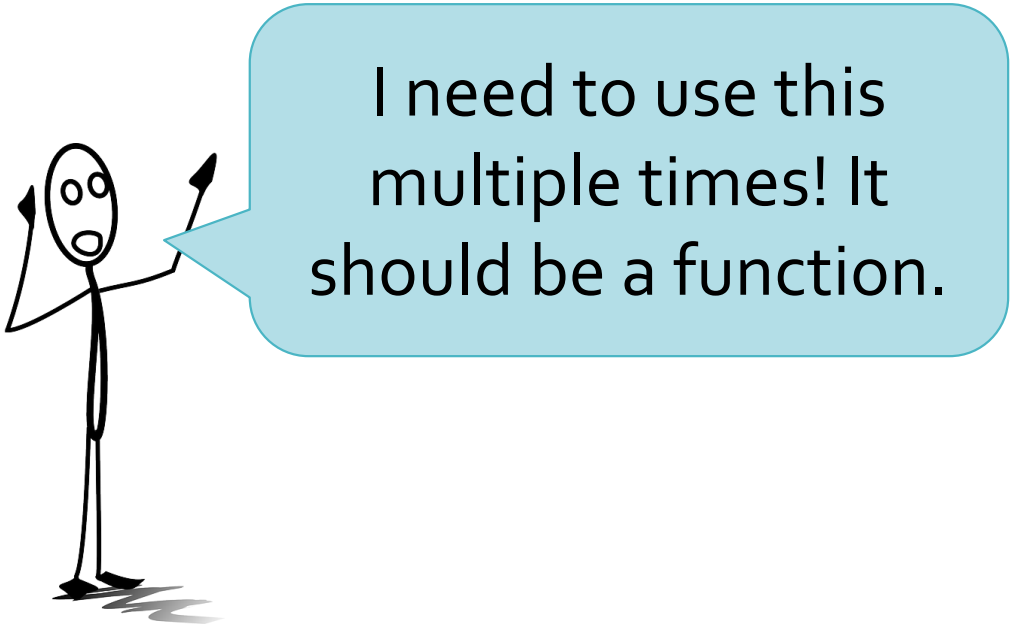
An Introduction to Computer Science

Defining Functions

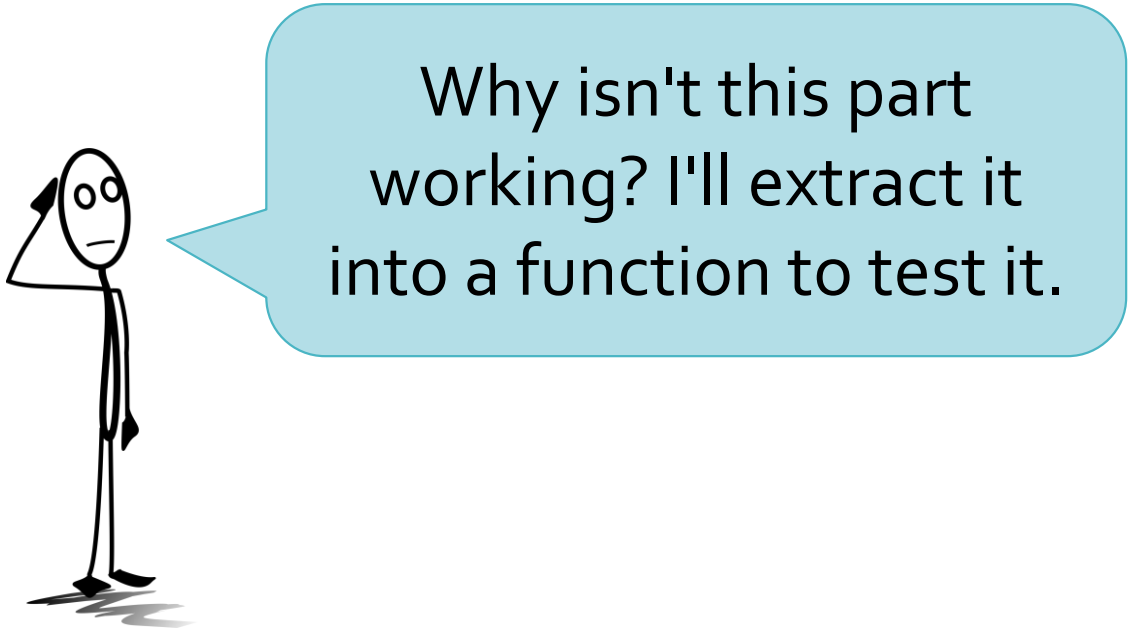


Why Functions?

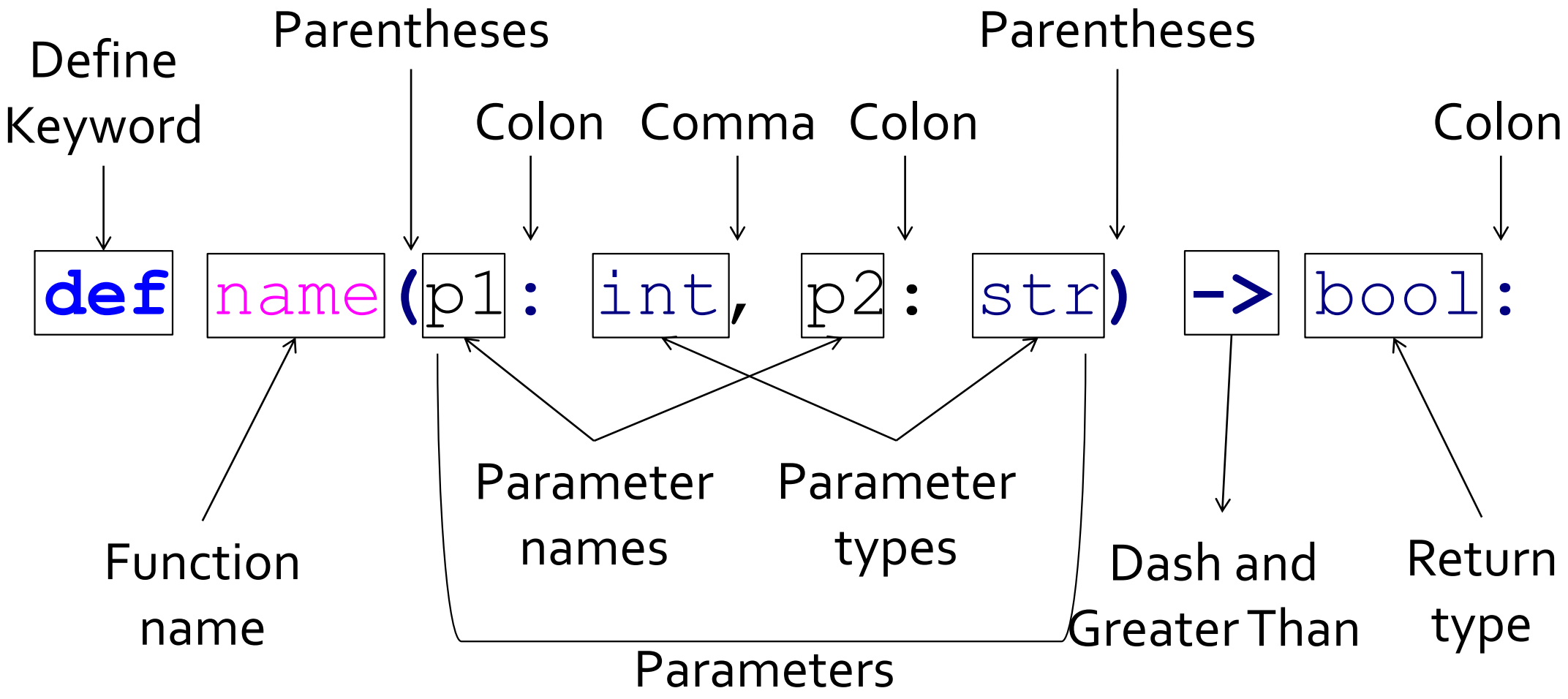
1. Code Reuse



2. Easier to debug



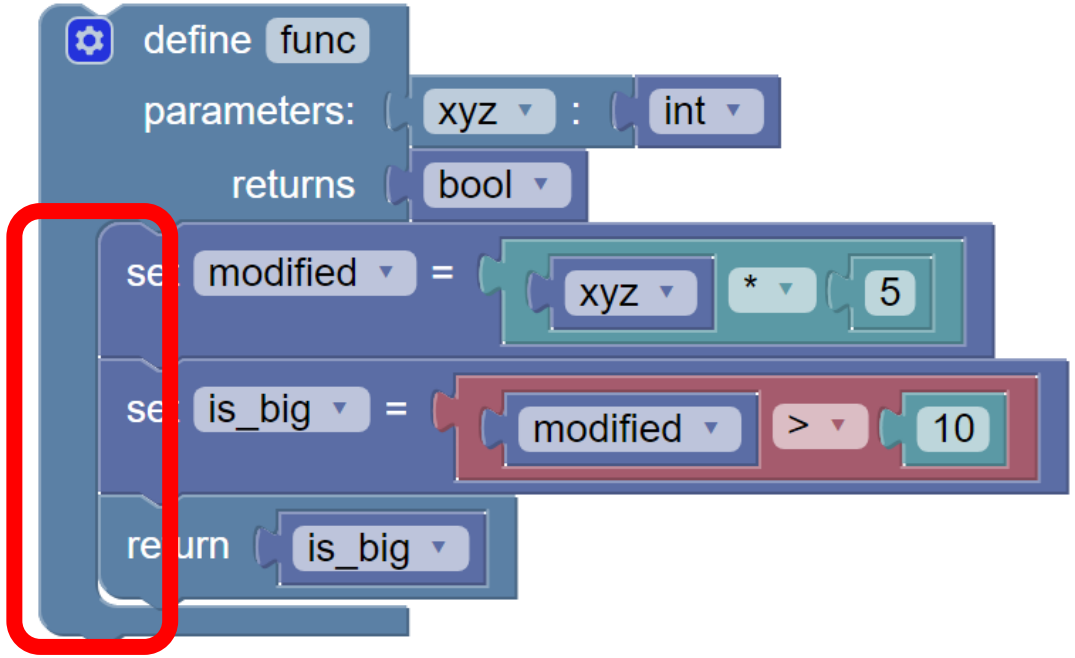
Definition Syntax



Function Body

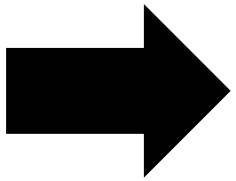
```
def func(xyz: int) -> bool:
    modified = xyz * 5
    is_big = modified > 10
    return is_big
```

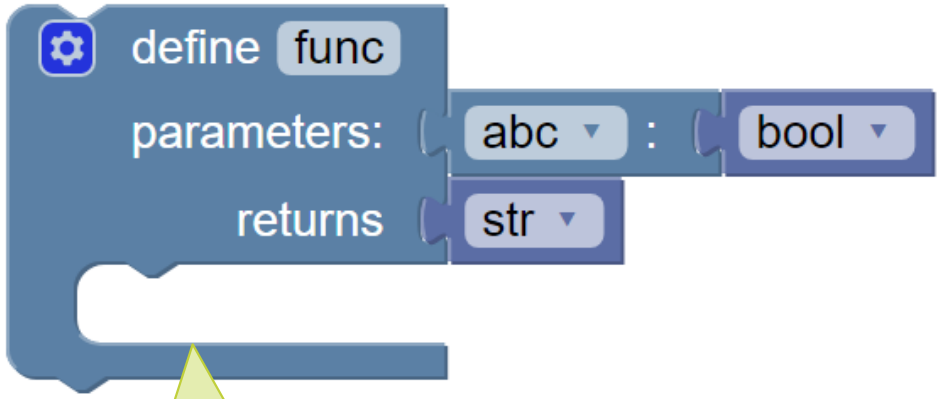
Indent with 4 spaces



Pass

```
def func(abc: bool)->str:
    pass
```





The "pass" means "do nothing"

Nothing there!

Tricky Colons

```
def name() -> None:
    pass
```

Seriously, don't forget the colon.

Naming a function

1. Names can only have
   - Letters
   - Numbers
   - Underscores (_)

2. Names must begin with
   - A letter
   - An underscore (_)

Calling Your Functions

```
def print_name(name: str):
    print("Hello", name)

print_name("Cory")
print_name("Klaus")
print_name("Ellie")
```

Parameters

```
def add5(a_number: int):
    new = a_number + 5

add5(10)
```

a_number will have the value 10

Parameters and Values

```
def reset(a_var: int):
    a_var = 0

my_num = 5
reset(my_num)
print(my_num)
```

This modifies a_var, but not my_num!!

So this prints 5, not 0!

Parameters and Types

Parameter type

Parameter type

```
def area(length:int, width:int) -> int:
    return length * width
```

```
area(4, 5)
```

These must match the parameter types!

Return

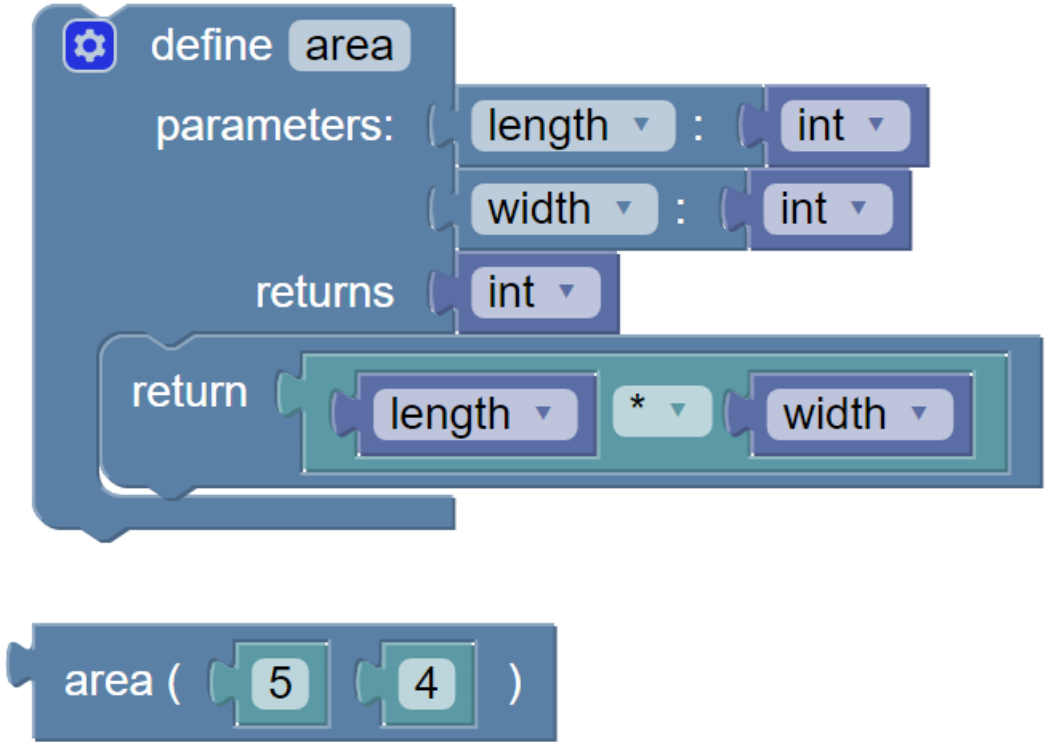
```
def area(length:int, width:int) -> int:
    return length * width
```

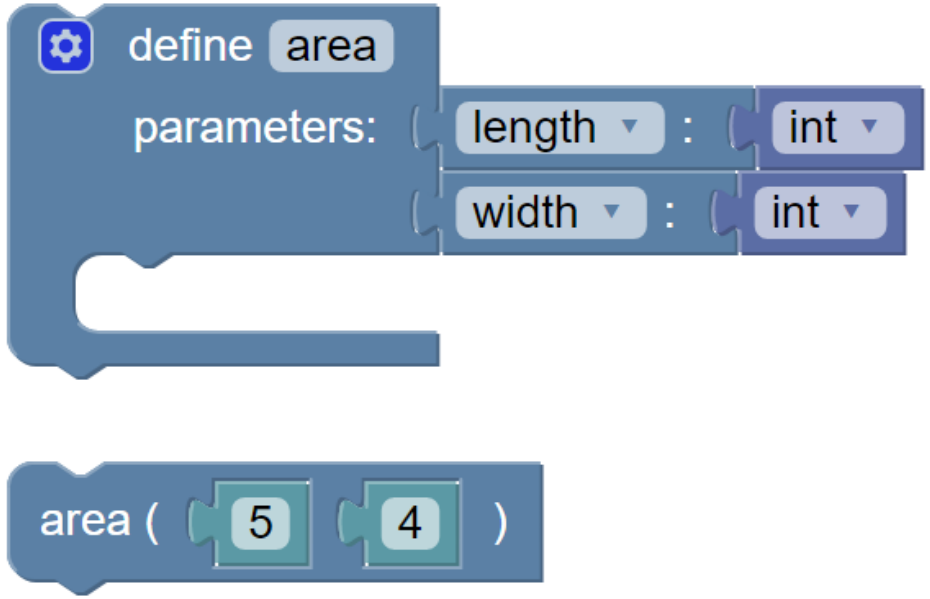
Return type

Arrow

Return statement

The Effect of Returning





Returns Go in Functions

Do not put return statements
OUTSIDE of a function body!

Print vs. Returning

```
def double(a_number:int)->int:
    return a_number * 2

doubled = double(3)
print(doubled)
```

Prints:
6

```
def double(a_number:int):
    print(a_number * 2)

doubled = double(3)
print(doubled)
```

Prints:
6
None