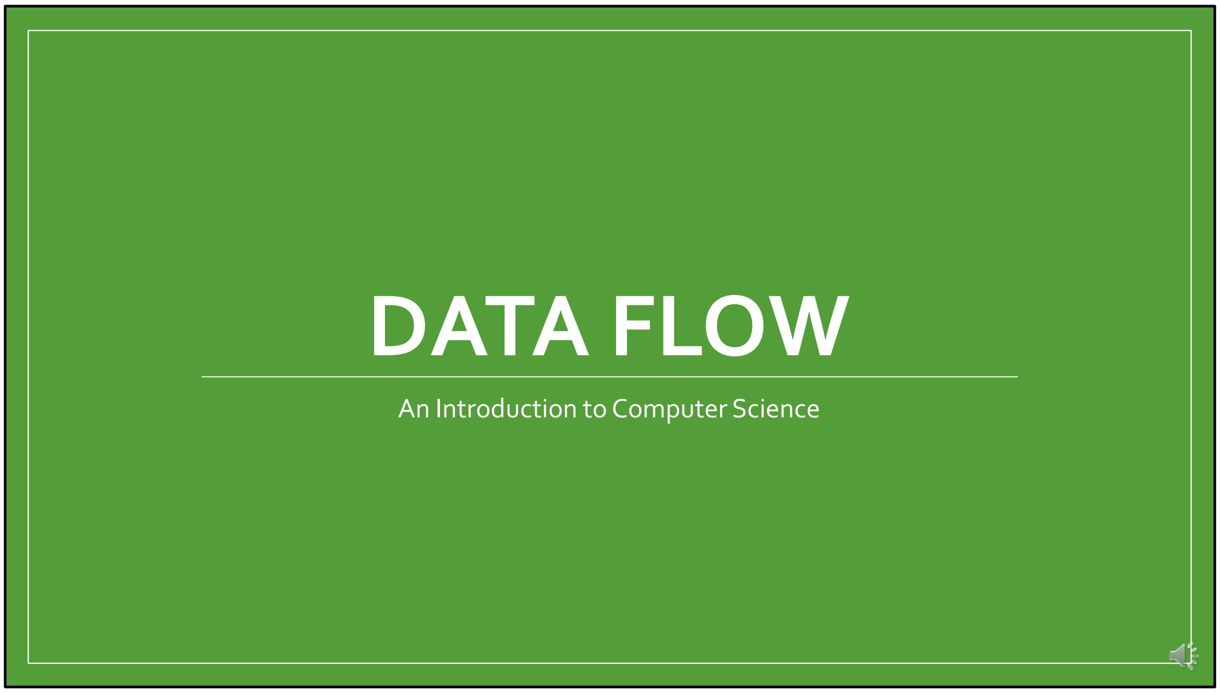
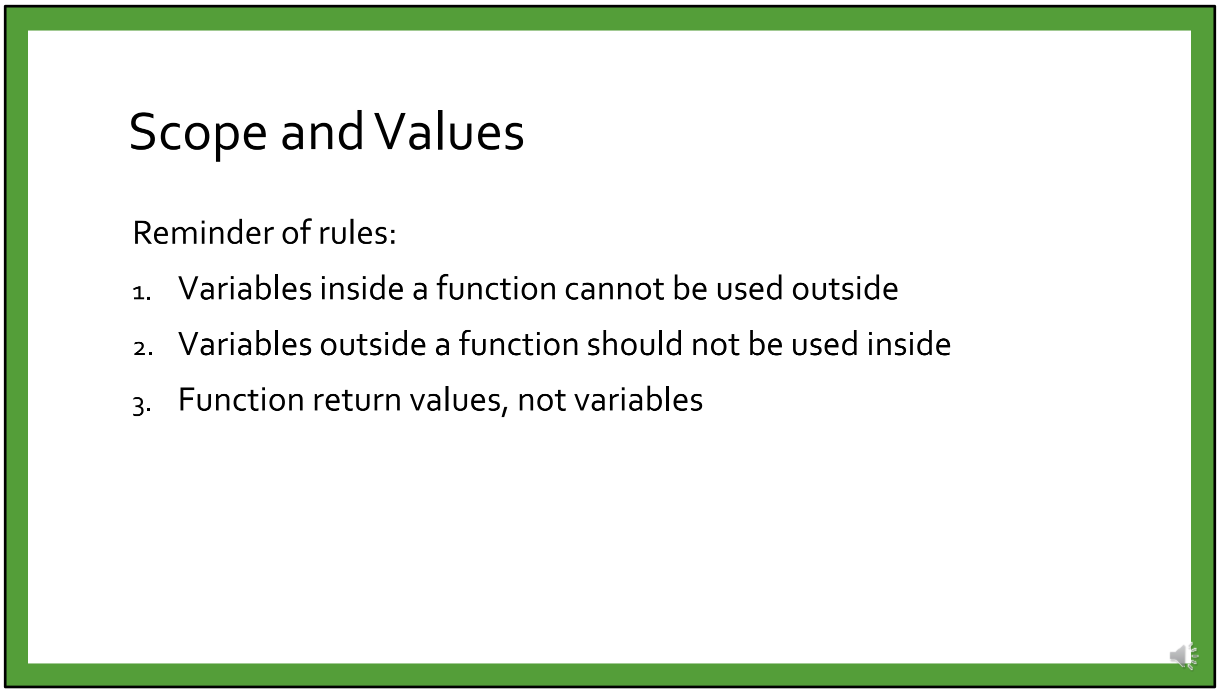


# DATA FLOW

An Introduction to Computer Science

Let's learn about Data Flow.

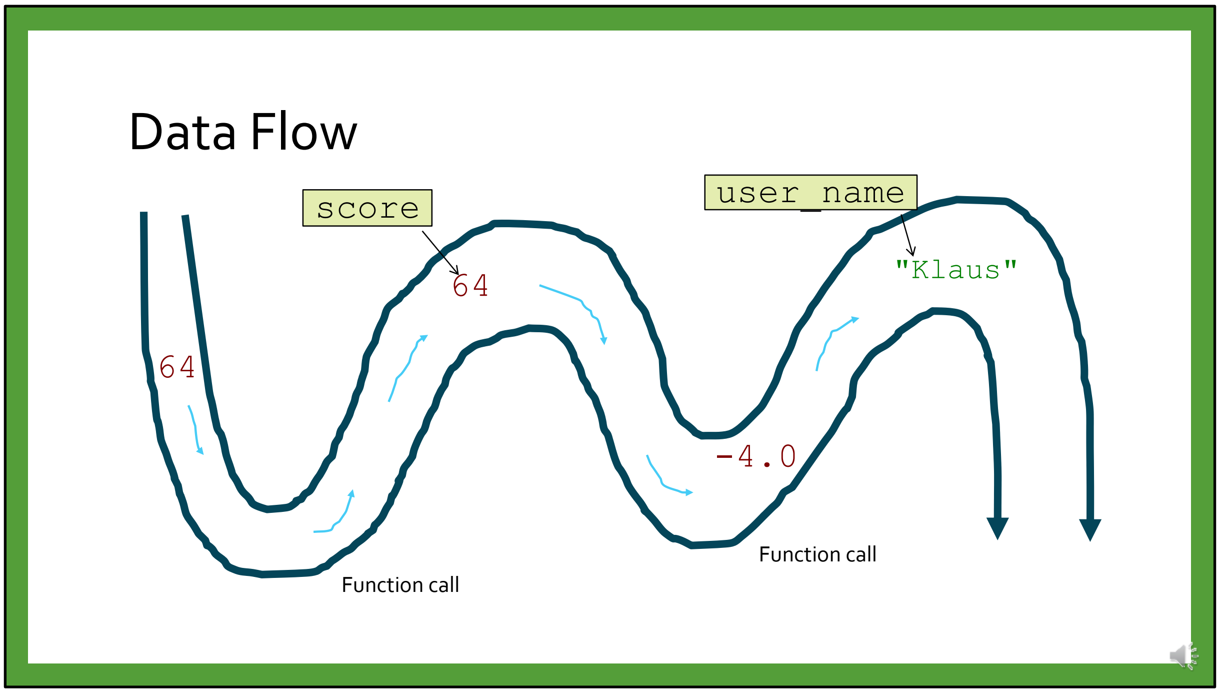


## Scope and Values

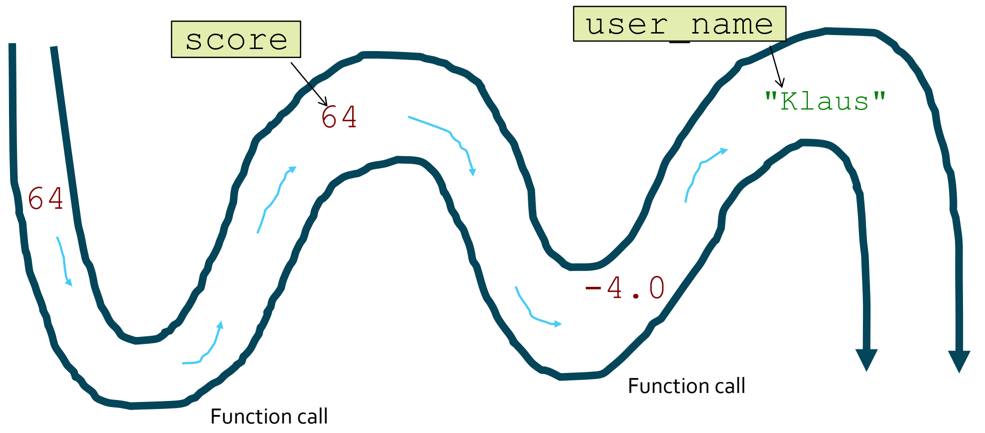
Reminder of rules:

1. Variables inside a function cannot be used outside
2. Variables outside a function should not be used inside
3. Function return values, not variables

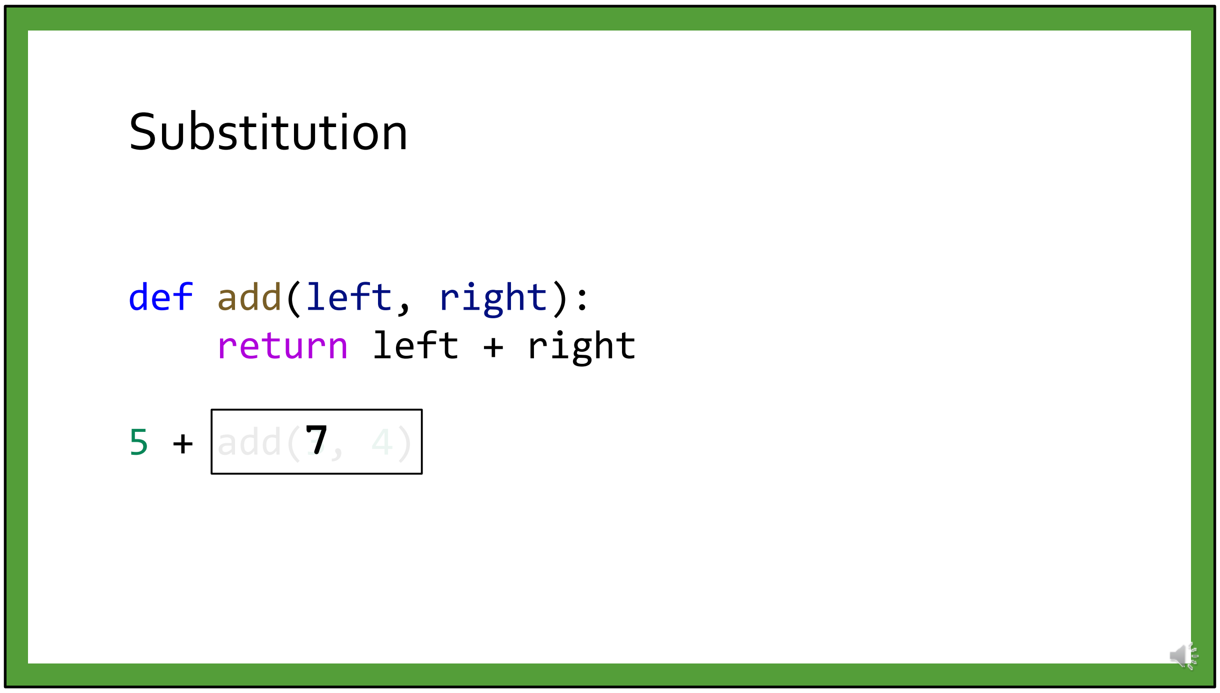
Previously, we learned about scope: the idea that variables inside a function cannot be used outside of the function, and variables outside a function should not be used inside the function.
Further, we learned that functions do not consume or return variables; instead they consume and return values.
These ideas are critical in understanding how data flows through a program.



## Data Flow



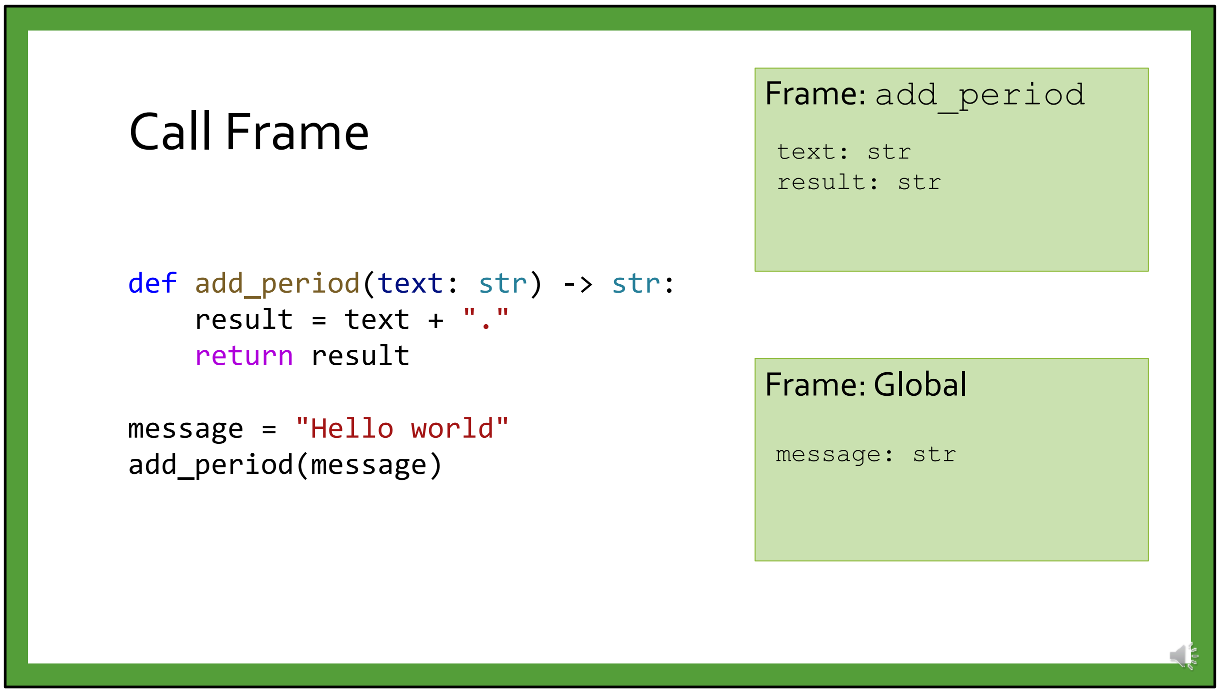
You can think of a program as a flowing, twisting river.
Regular execution makes the river flow south, but functions disrupt this flow.
Along the way, values are carried by the current.
At times, we give these values names by using variables, but it is the values that flow through the program, not the variables.



## Substitution

```
def add(left, right):
    return left + right

5 + add(7, 4)
```

When you call a function, each parameter is assigned the value of a relevant argument.
When you return from a function, the function call is substituted for the returned value.
These are the only tools we should use in Python to move data around a program.
This is just like what happens when you use an operator like plus or minus.
Even though we do not see the substitution visually, it still happens.



## Call Frame

```python
def add_period(text: str) -> str:
    result = text + "."
    return result

message = "Hello world"
add_period(message)
```

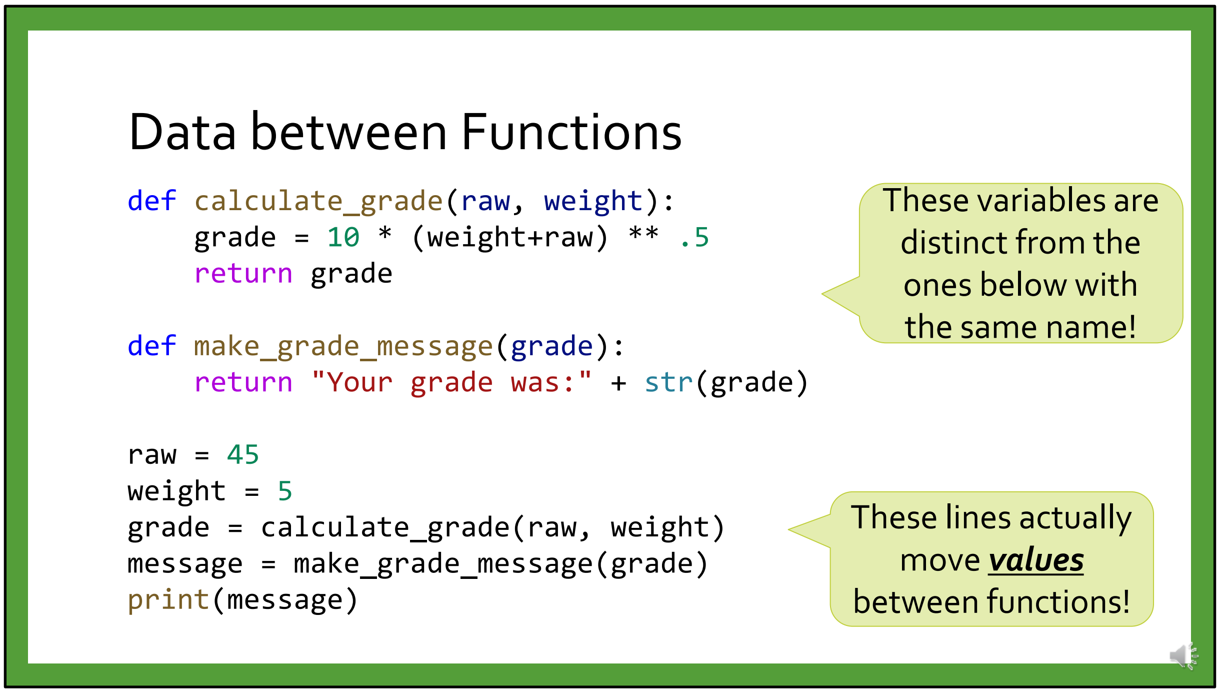
Frame: add_period

```
text: str
result: str
```

Frame: Global

```
message: str
```

Each time we call a function, we say that its local variables are i
nside their
own scope.
We will also call this scope the **frame**, and it can be used to s
how
the current variables and their values.
Notice that the global frame is separate from the local frame.



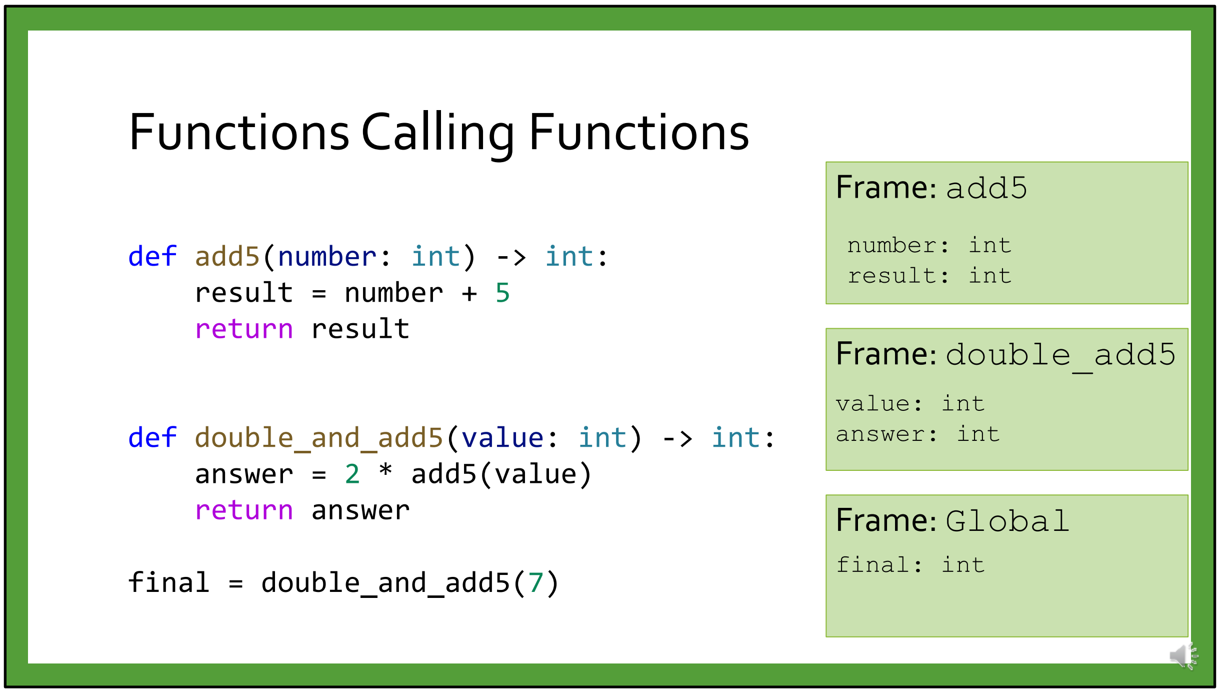
## Data between Functions

```python
def calculate_grade(raw, weight):
    grade = 10 * (weight+raw) ** .5
    return grade

def make_grade_message(grade):
    return "Your grade was:" + str(grade)

raw = 45
weight = 5
grade = calculate_grade(raw, weight)
message = make_grade_message(grade)
print(message)
```

These variables are distinct from the ones below with the same name!

These lines actually move ***values*** between functions!

To move data from one function to another, you cannot just look at the two functions.
You must also look at where the functions were called.
The returned value of one function should be fed into the next function.



## Functions Calling Functions

```python
def add5(number: int) -> int:
    result = number + 5
    return result


def double_and_add5(value: int) -> int:
    answer = 2 * add5(value)
    return answer

final = double_and_add5(7)
```

Frame: add5

```
number: int
result: int
```

Frame: double_add5

```
value: int
answer: int
```

Frame: Global

```
final: int
```

Things get even more complicated when functions call other function
s, but
this happens all the time.
We are used to calling built-
in functions, but we can call our own functions too.
When we call a function inside another function, we **stack** the n
ew function
call's frame on top of the current frame, with the top frame being
the current scope.
When a function call ends, we remove that top stack and return to t
he one below it.
Each function call's frame is separate from all other frames, even
if they
come from the same function.