USABLE PROGRAMMING TOOLS FOR EXPERIMENTAL BIOLOGISTS

Justin Lubin

Advisor: Sarah E. Chasins

EPIC Retreat, Spring 2022

Vaccines

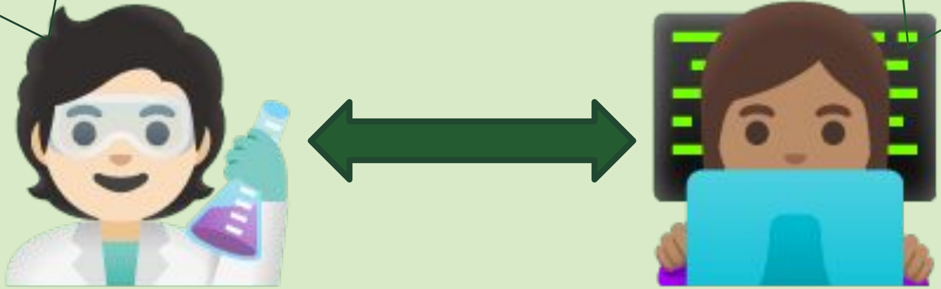
Precision health

Genomic editing

Wet lab ↔ Dry lab

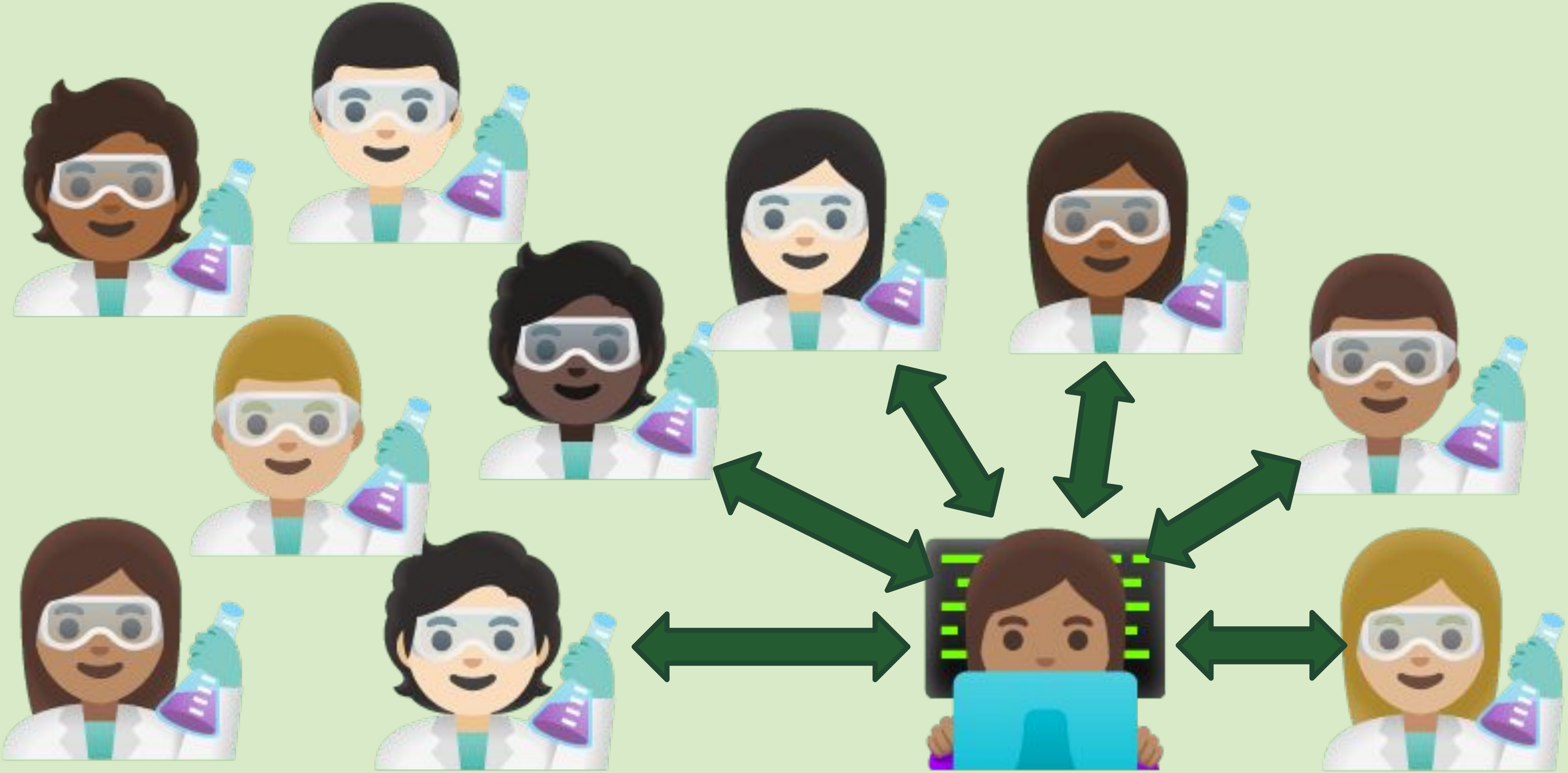
HANDS-ON PROTOCOLS

CODE, MATH

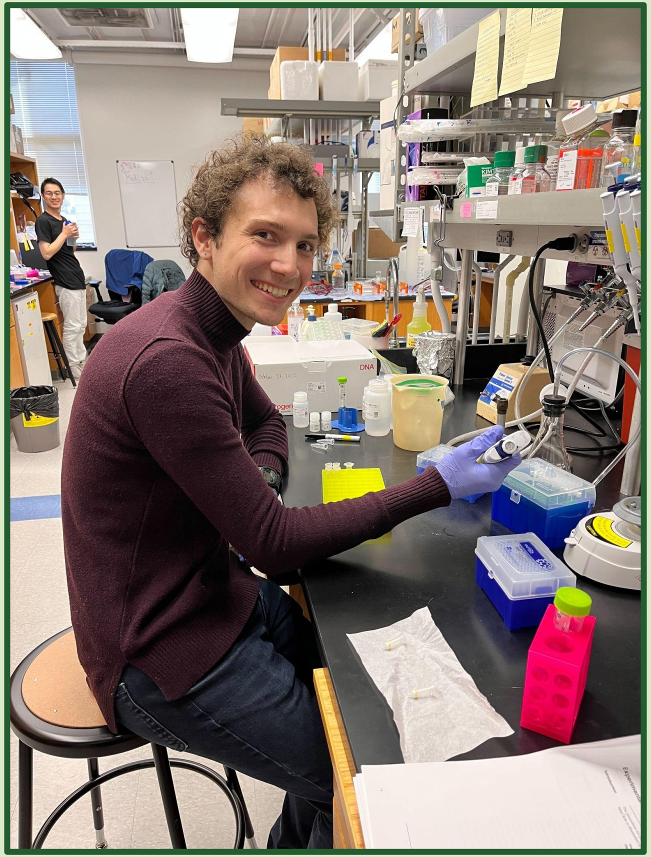


Wet lab

Dry lab



NUÑEZ lab
EXIT


DNA

USABLE PROGRAMMING TOOLS FOR EXPERIMENTAL BIOLOGISTS

Vaccines

💉

Precision health

🎯

Genomic editing

🧬

COMPONENT-BASED REFACTORING

WET LAB LANGUAGE

COMPONENT-BASED REFACTORING

WET LAB
LANGUAGE

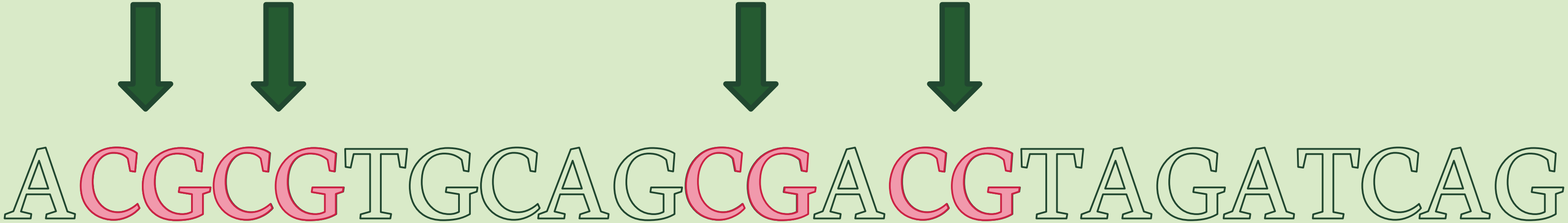
COMPONENT-BASED REFACTORING

1. Motivating example
2. Naïve approach
3. Our approach

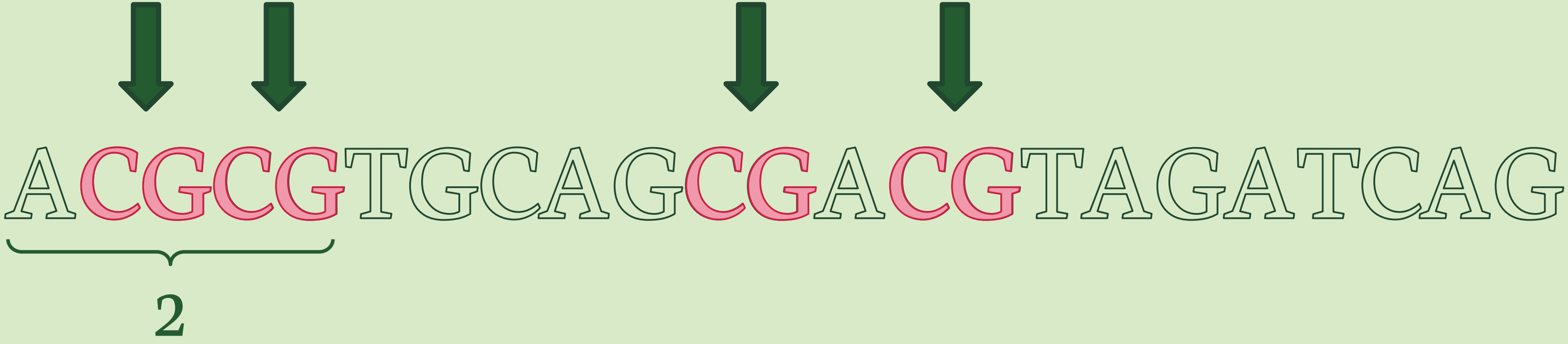
COMPONENT-BASED REFACTORING

1. **Motivating example**
2. Naïve approach
3. Our approach

ACGCGTGCAGCGACGTAGATCAG

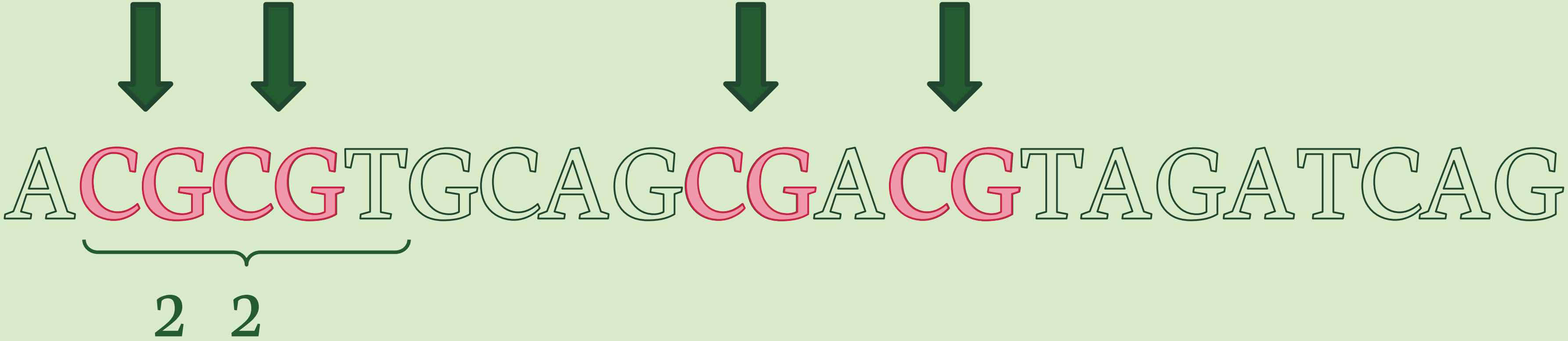


A**CGCG**TGCAG**CG**A**CG**TAGATCAG



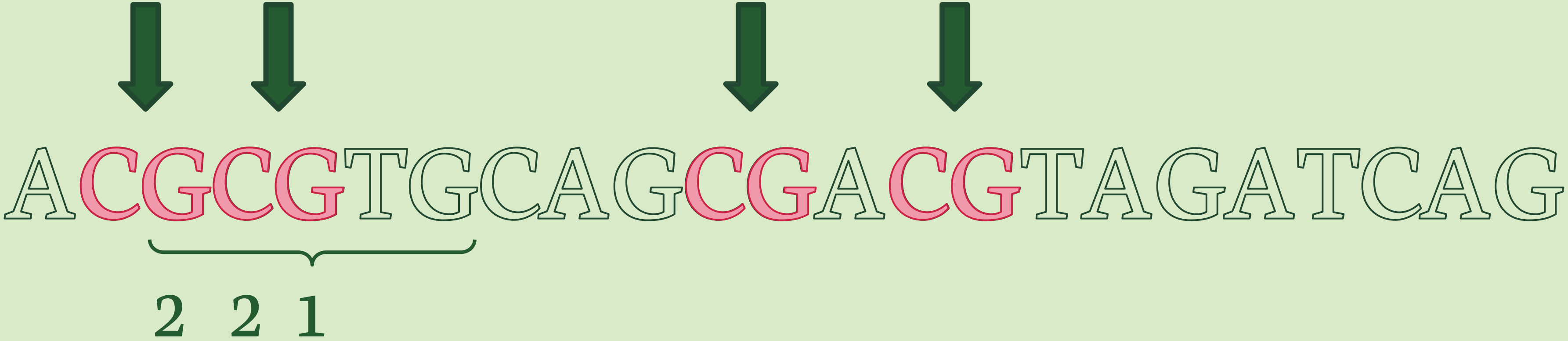
ACGCGTGCAGCGACGTAGATCAG

2



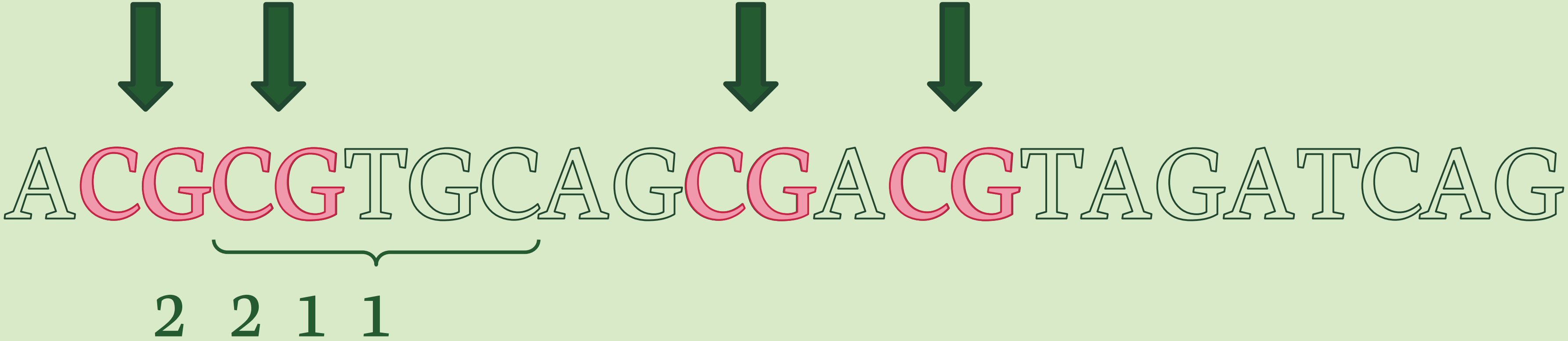
ACGCGTGCAGCGACGTAGATCAG

2 2



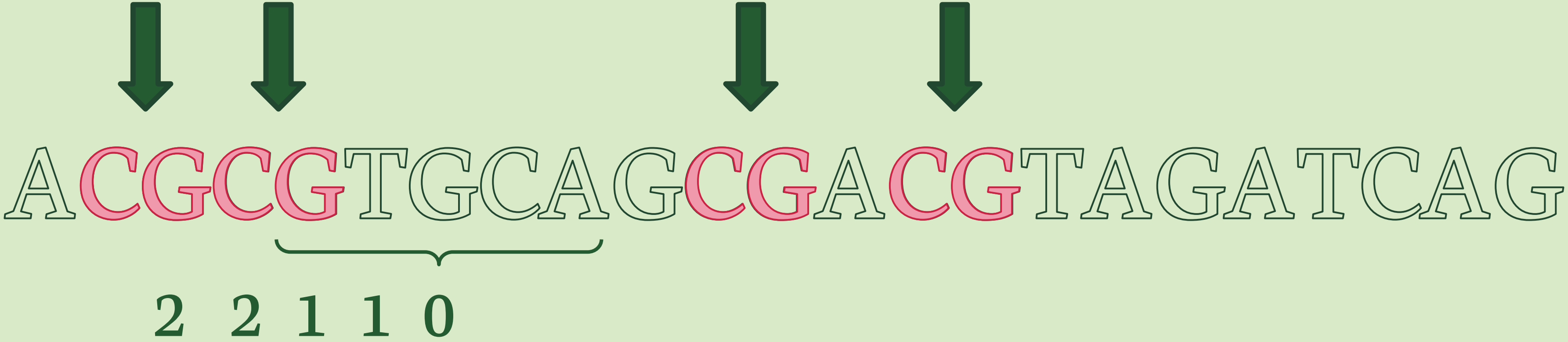
ACGCGTGCAGCGACGTAGATCAG

2 2 1



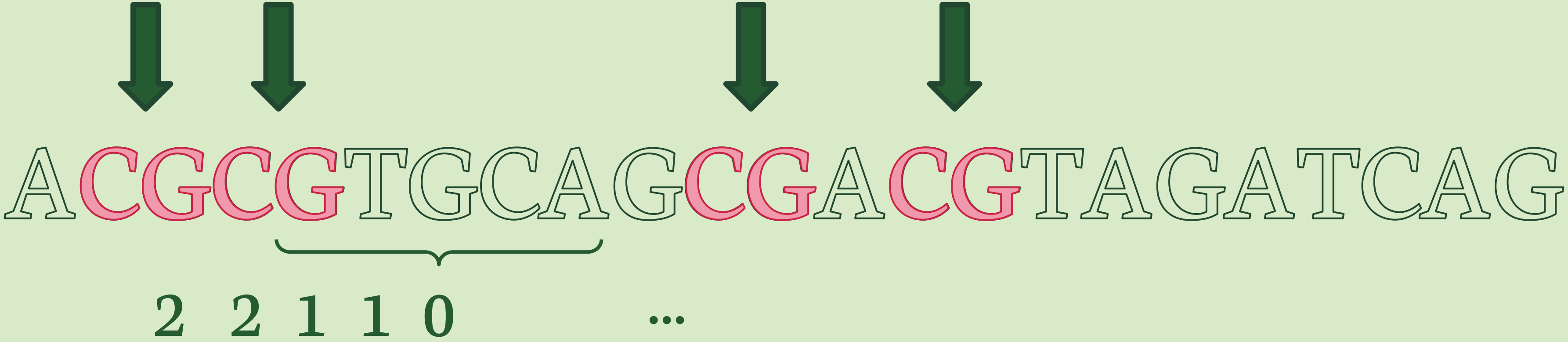
ACGCGTGCAGCGACGTAGATCAG

2 2 1 1



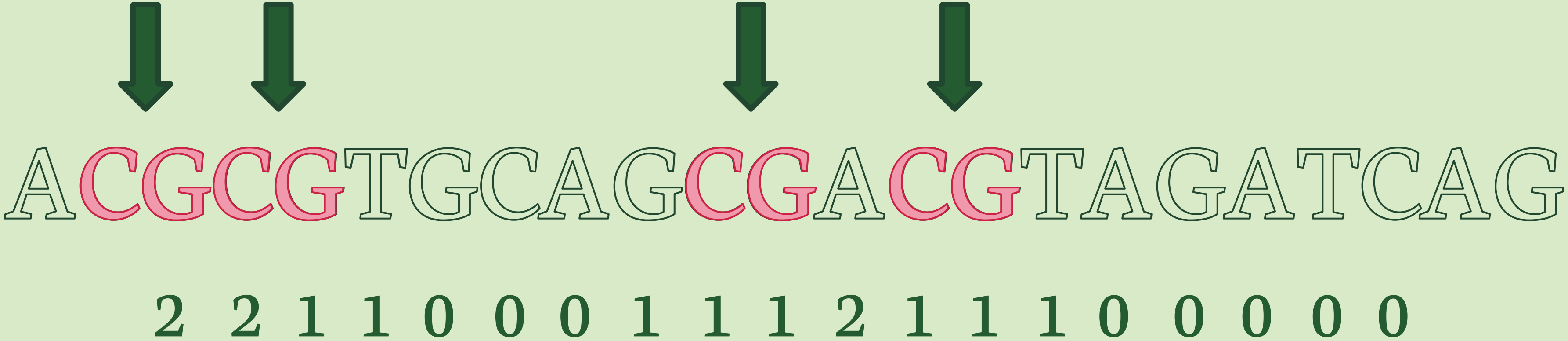
ACGCGTGCAGCGACGTAGATCAG

2 2 1 1 0



ACGCGTGCAGCGACGTAGATCAG

2 2 1 1 0 …



ACGCGTGCAGCGACGTAGATCAG

2 2 1 1 0 0 0 1 1 1 2 1 1 1 0 0 0 0 0

Straightforward solution:

Two nested `for` loops

*...but takes **>1 day** to run...*

Faster solution:

```
np.concatenate([
  np.convolve(
    (seq == "C")[:-1] & (seq == "G")[1:],
    np.ones(window_size),
    "valid"),
  [0]])
```

Straightforward solution:

Two nested `for` loops

*…but takes **>1 day** to run…*

Faster solution:

```
np.concatenate([
  np.convolve(
    (seq == "C")[:-1] & (seq == "G")[1:],
    np.ones(window_size),
    "valid"),
  [0]])
```

*…but takes **<15 min** to run!*

COMPONENT-BASED REFACTORING

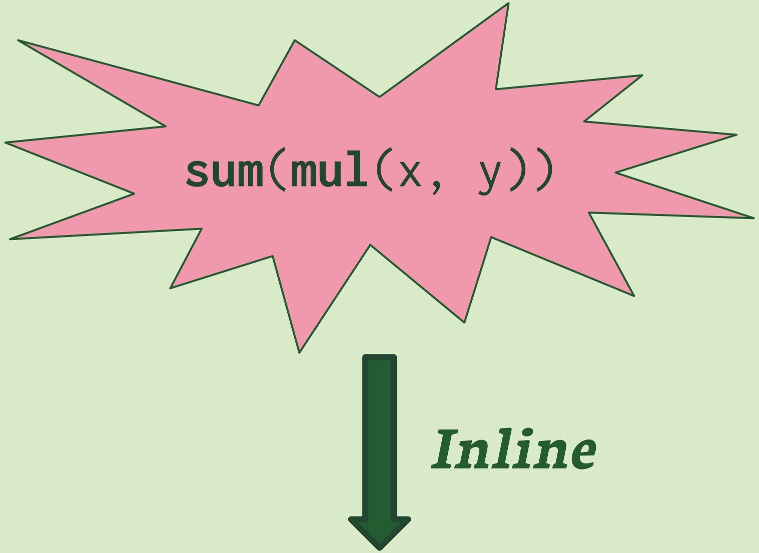
1. **Motivating example**
2. Naïve approach
3. Our approach

COMPONENT-BASED REFACTORING

1. Motivating example
2. **<u>Naïve approach</u>**
3. Our approach

```
sum(mul(x, y))
```

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

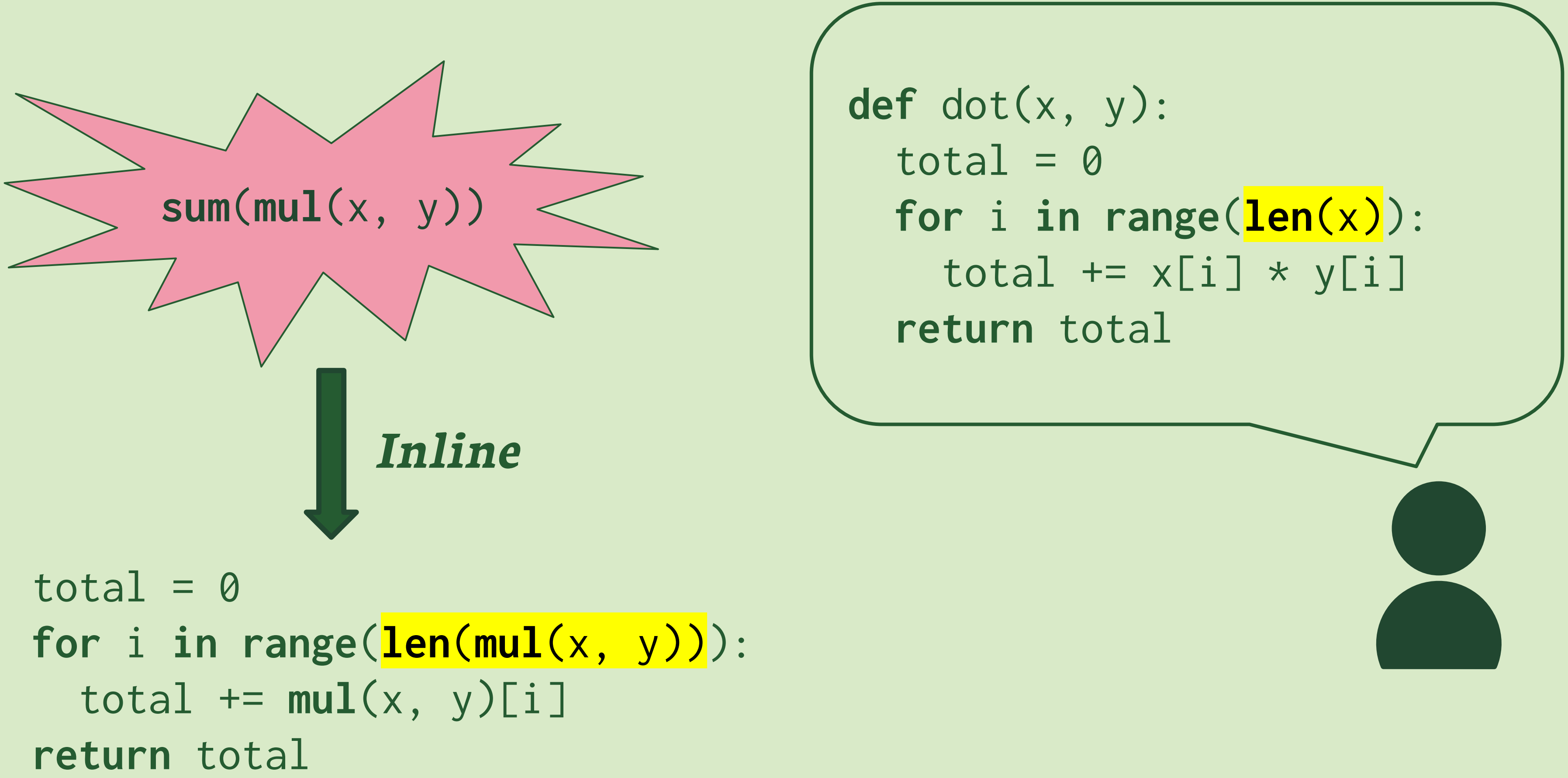


```
sum(mul(x, y))
```

Inline

```
total = 0
for i in range(len(mul(x, y))):
  total += mul(x, y)[i]
return total
```

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

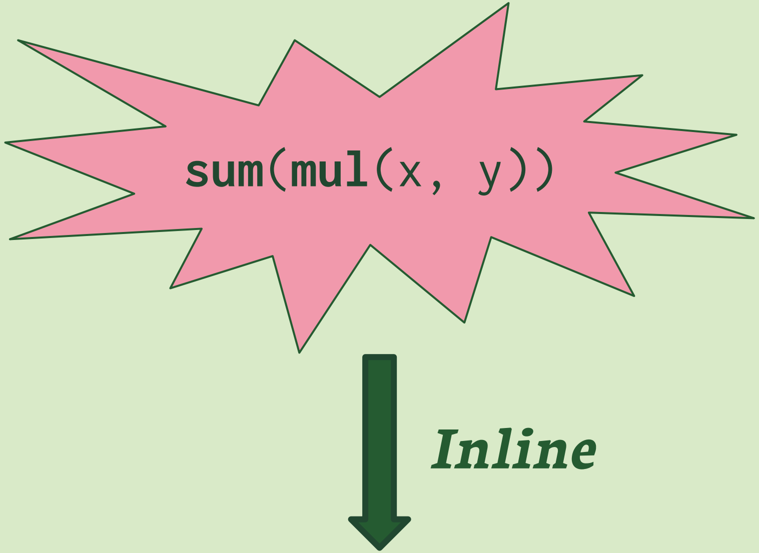


```
sum(mul(x, y))
```

Inline

```
total = 0
for i in range(len(mul(x, y))):
  total += mul(x, y)[i]
return total
```

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```



```
total = 0
for i in range(len(mul(x, y))):
  total += mul(x, y)[i]
return total
```

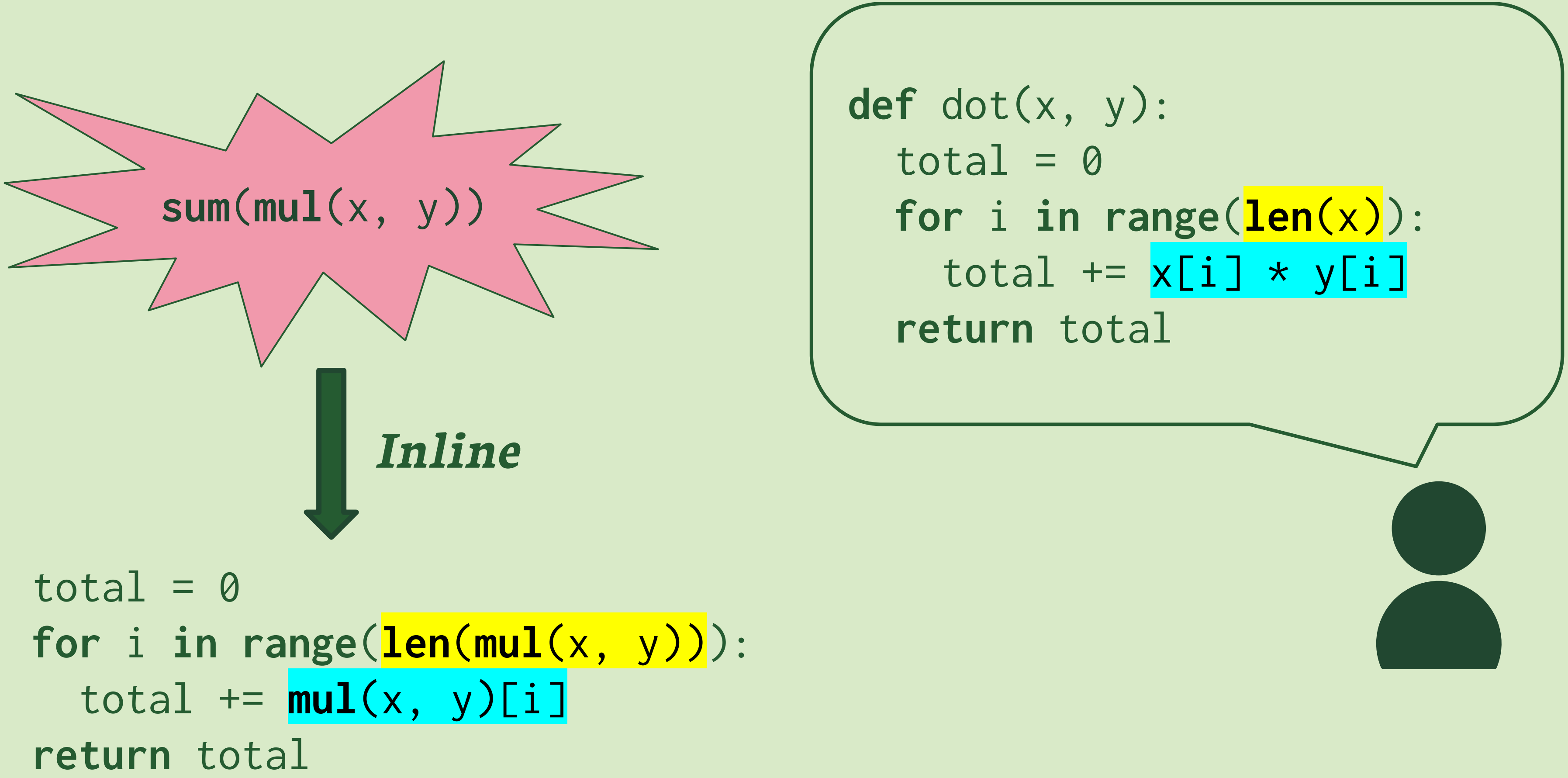
```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

COMPONENT-BASED REFACTORING

1. Motivating example
2. **<u>Naïve approach</u>**
3. Our approach

COMPONENT-BASED REFACTORING

1. Motivating example
2. Naïve approach
3. **Our approach**



sum(mul(x, y))

Inline

```
total = 0
for i in range(len(mul(x, y))):
  total += mul(x, y)[i]
return total
```

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

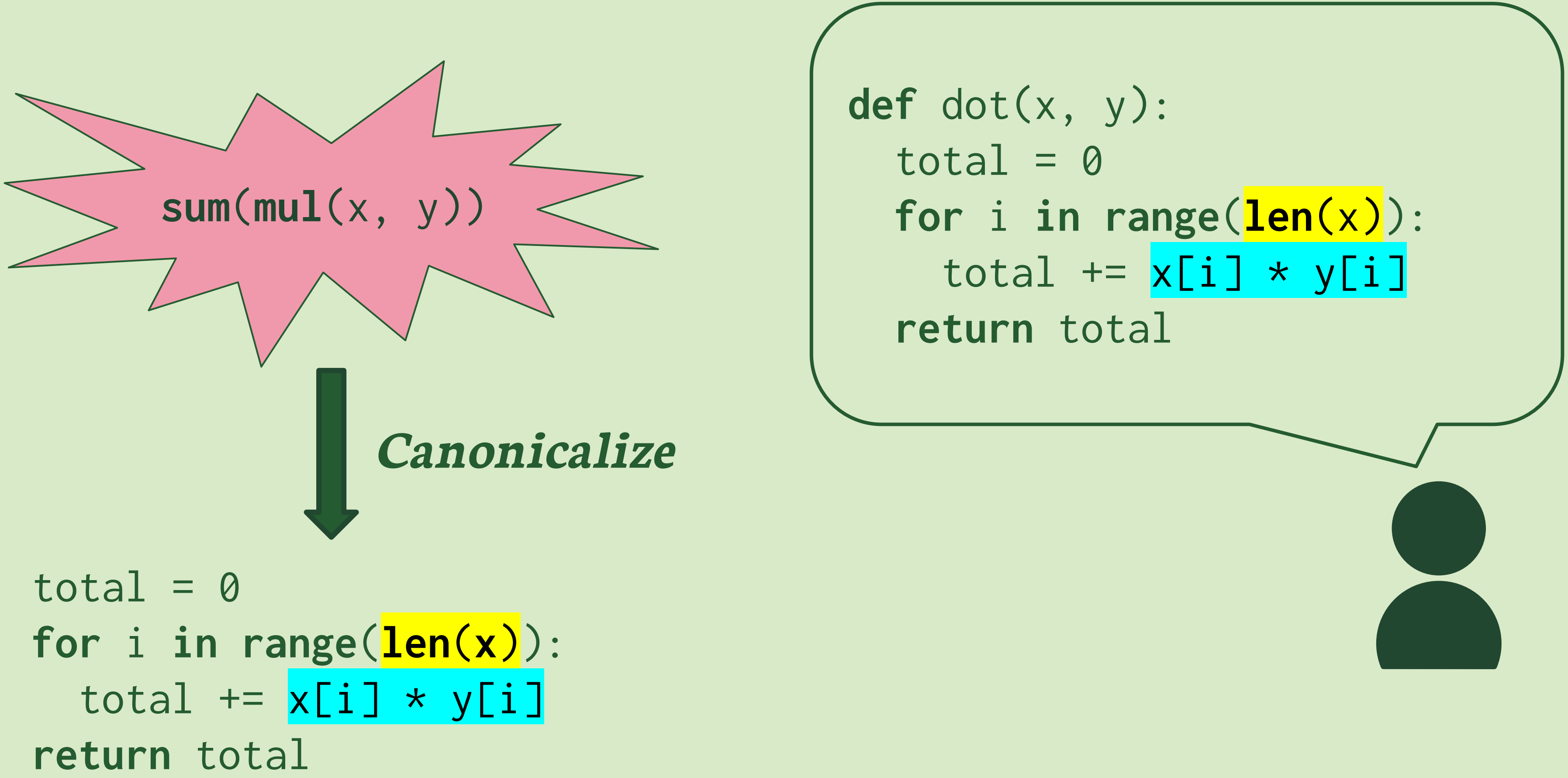
```
    len(x)
x[i] * y[i]
```

```
    len(mul(x, y))
mul(x, y)[i]
```

CANONICALIZATION

`len(mul(x, y))` → `len(x)`

`mul(x, y)[i]` → `x[i] * y[i]`



```
sum(mul(x, y))
```

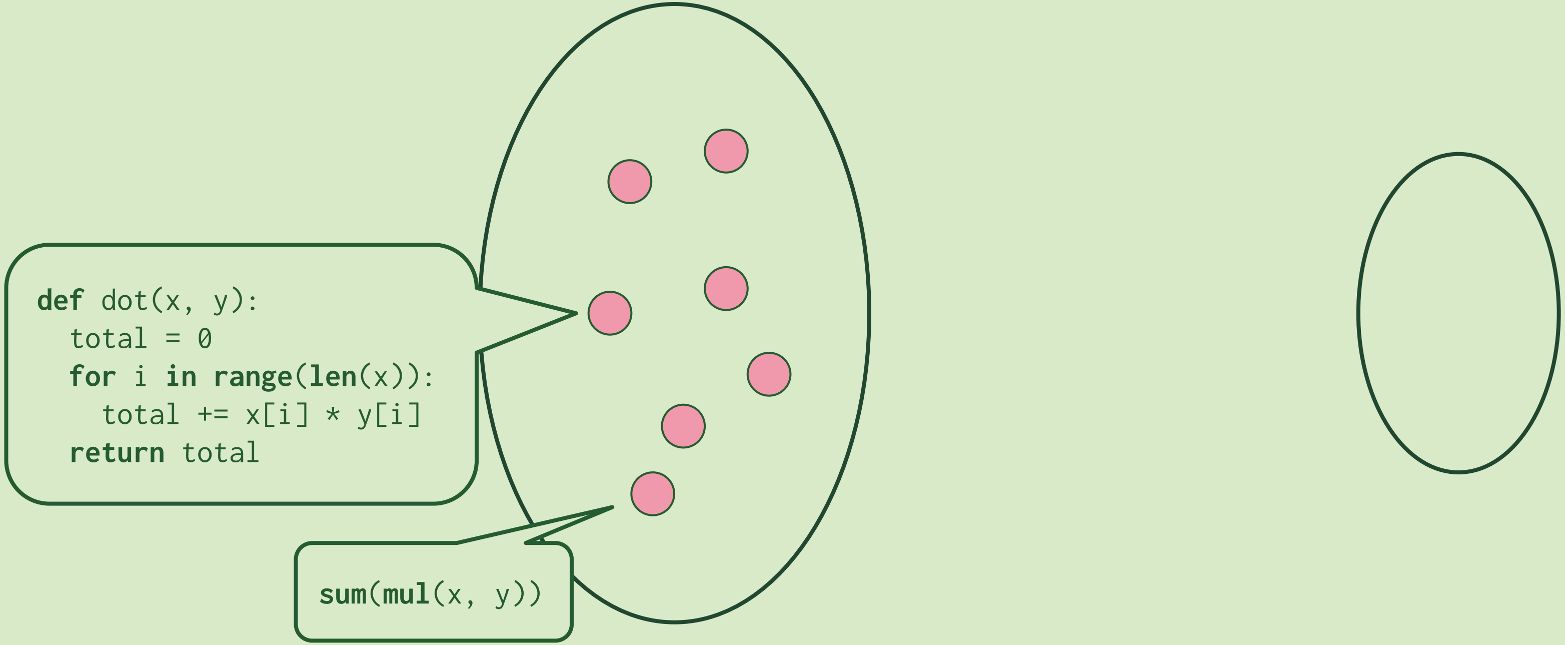
Canonicalize

```
total = 0
for i in range(len(x)):
  total += x[i] * y[i]
return total
```

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

↓ ***Canonicalize***

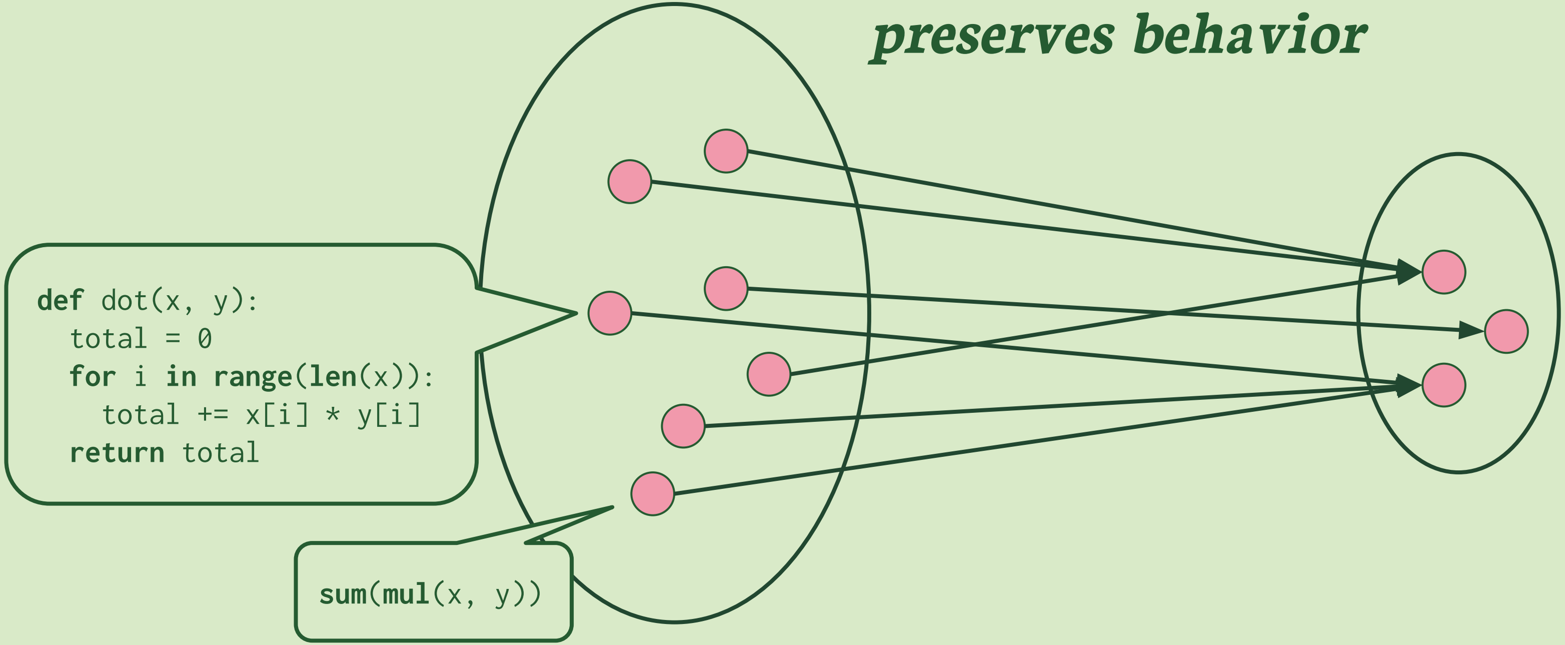
Canonicalize



```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

Canonicalize

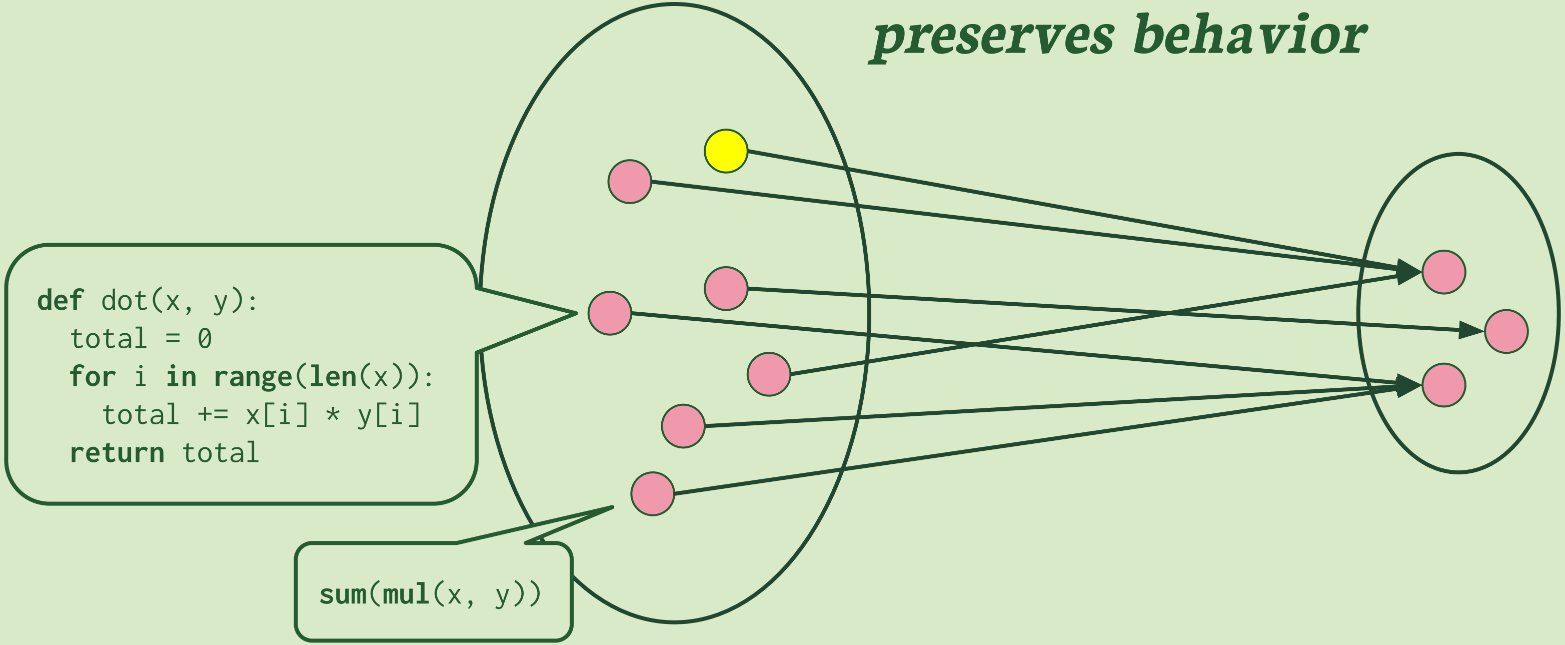


preserves behavior

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

Canonicalize

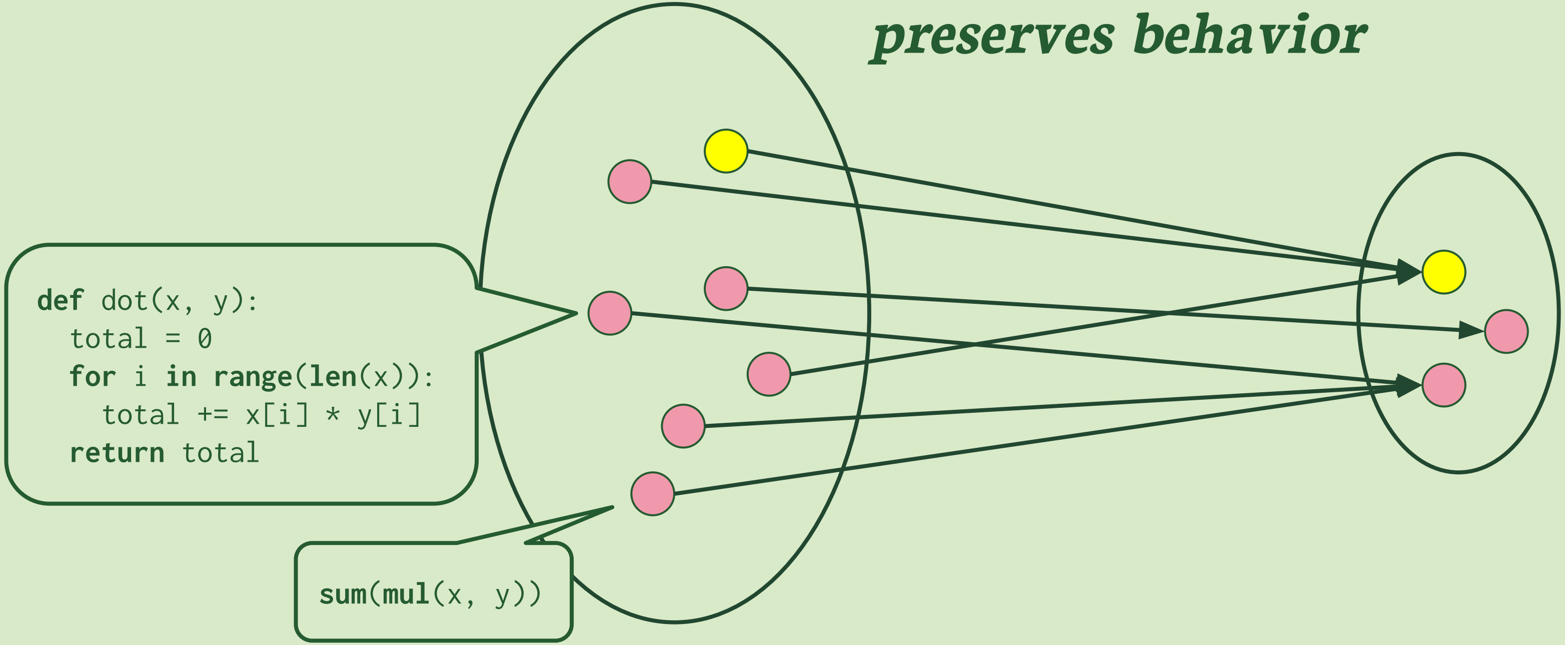


preserves behavior

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

Canonicalize

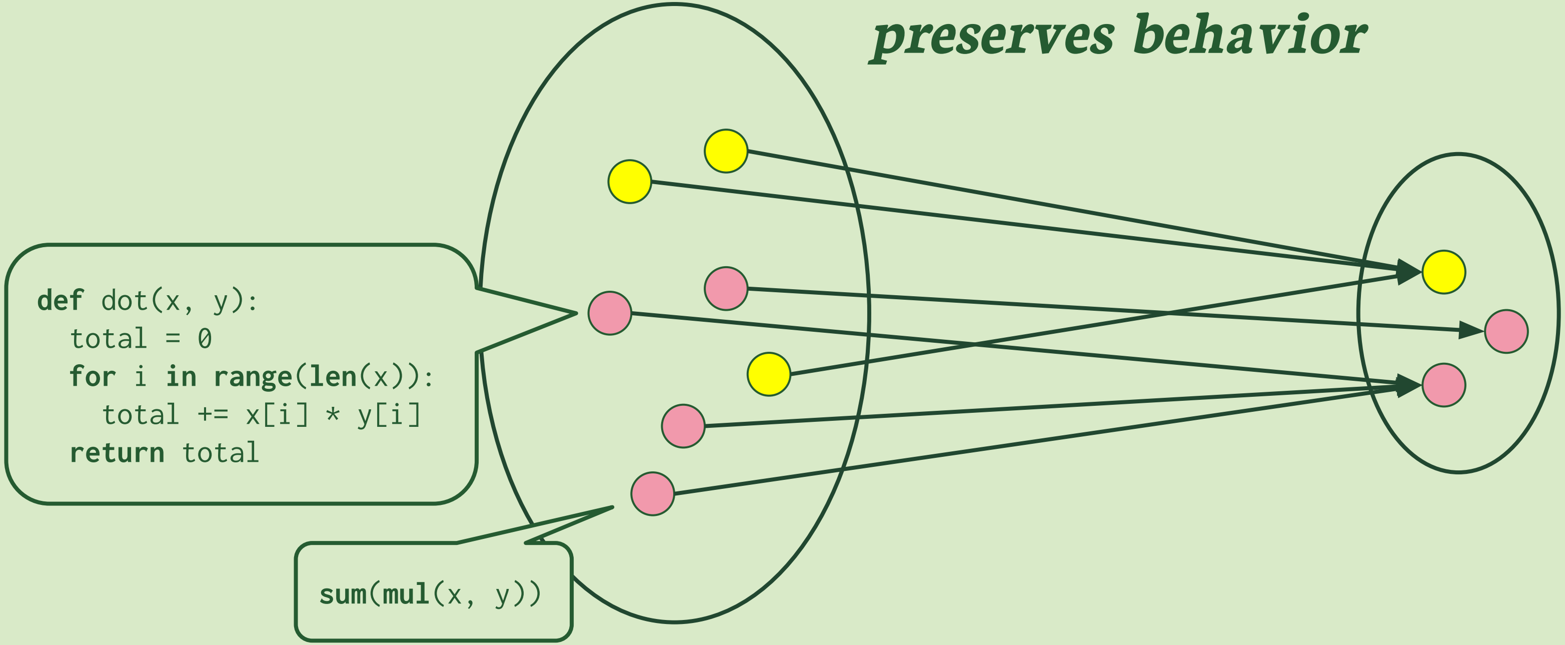


preserves behavior

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

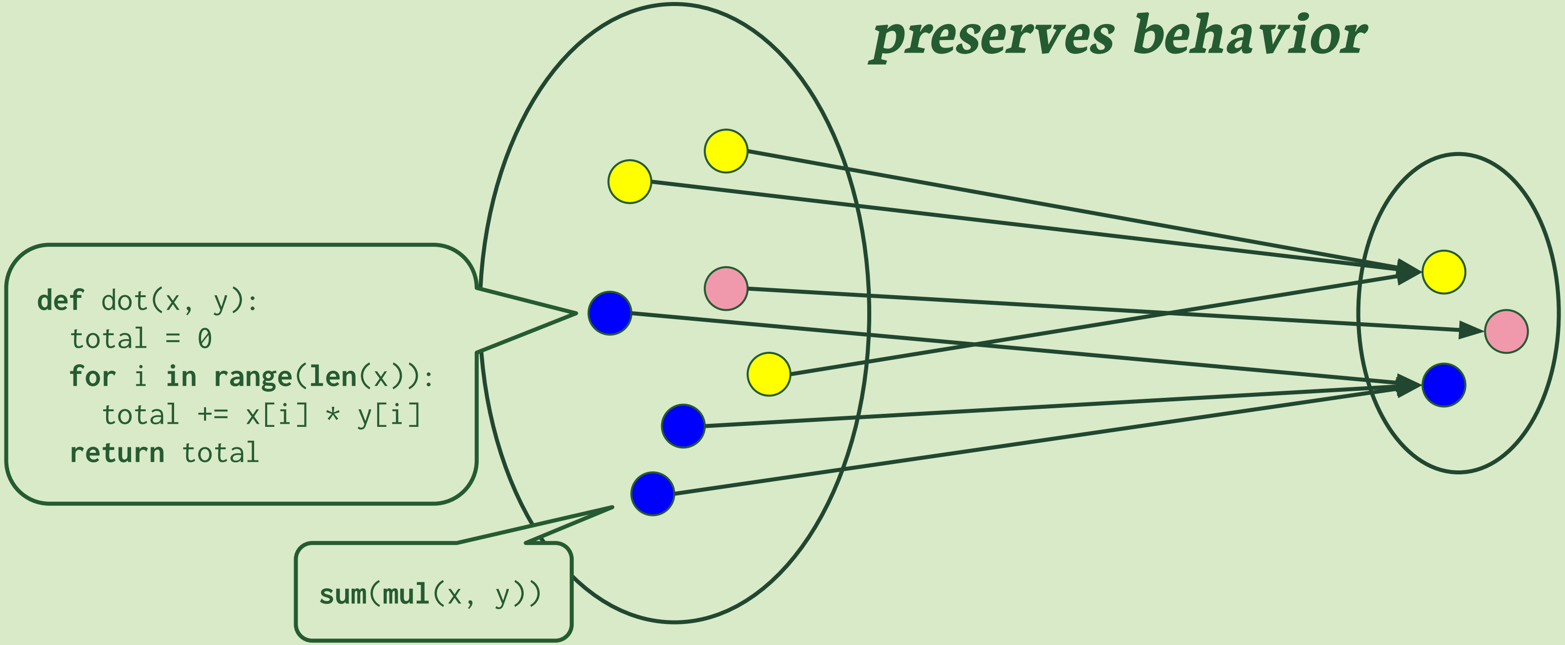
Canonicalize



```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

Canonicalize



preserves behavior

```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```

```
sum(mul(x, y))
```

KEY IDEA

For many programs,

KEY IDEA

For many programs,

we can check ***extensional equality***

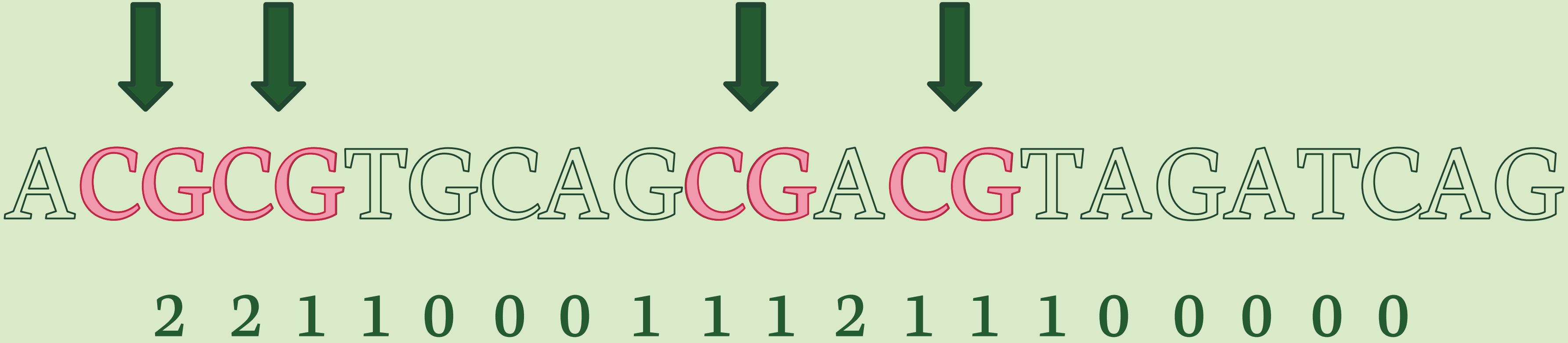
KEY IDEA

For many programs,

we can check
extensional equality

– by –

syntactic equality
modulo
canonicalization



ACGCGTGCAGCGACGTAGATCAG

2 2 1 1 0 0 0 1 1 1 2 1 1 1 0 0 0 0 0

Straightforward solution:

Two nested `for` loops

*...takes **>1 day** to run...*

Faster solution:

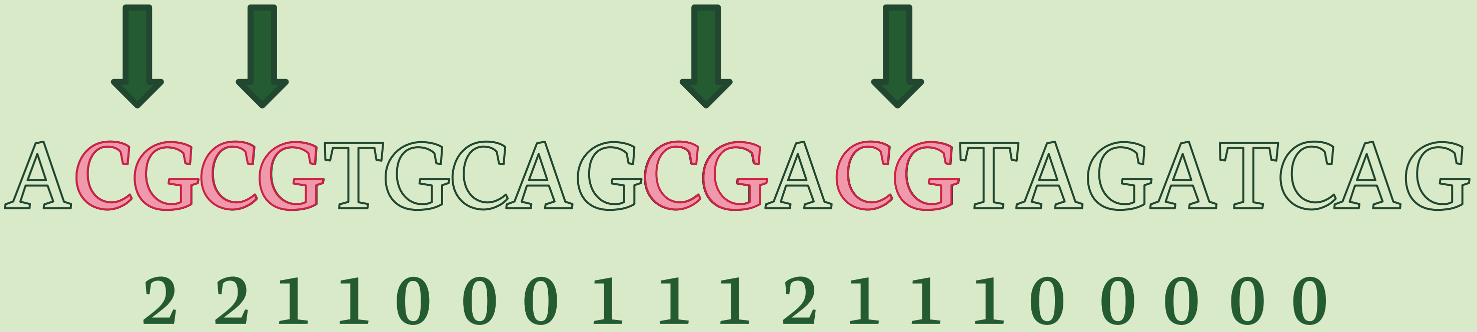
```
np.concatenate([
  np.convolve(
    (seq == "C")[:-1] & (seq == "G")[1:],
    np.ones(window_size),
    "valid"),
  [0]])
```

*...takes **<15 min** to run!*

Jeremy Ferguson

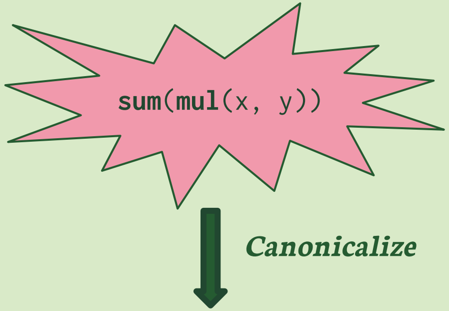
Kevin Ye

Jacob Yim



ACGCGTGCAGCGACGTAGATCAG

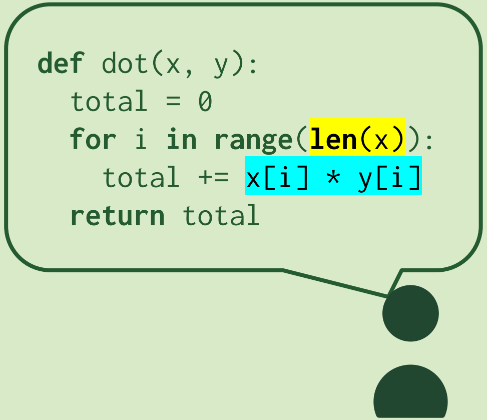
2 2 1 1 0 0 0 1 1 1 2 1 1 1 0 0 0 0 0



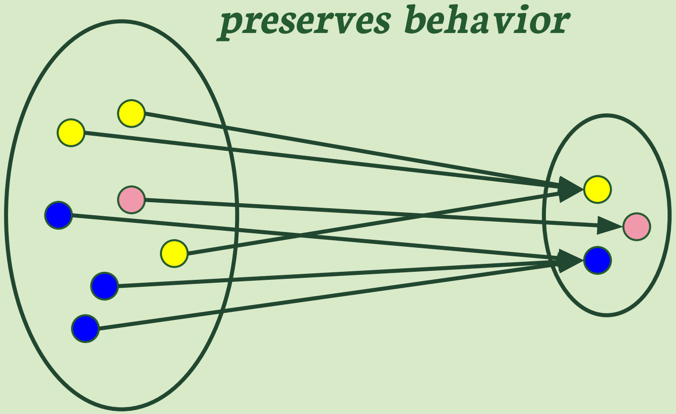
```
sum(mul(x, y))
```

Canonicalize

```
total = 0
for i in range(len(x)):
  total += x[i] * y[i]
return total
```



```
def dot(x, y):
  total = 0
  for i in range(len(x)):
    total += x[i] * y[i]
  return total
```


preserves behavior