Short-circuit Evaluation

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Logical Operators

and

Logical conjunction

```
operand_a and operand_b
```

True if, and only if,
both operands are true

or

Logical disjunction

```
operand_a or operand_b
```

True if either or both operands are true

False if, and only if,
both operands are false

What Is True

Truthy

Objects for which `bool(obj)` returns `True`

Non-zero numbers: `42`, `True`, `3.142`

Non-empty collections: `[1, 2, 3]`, `"Not empty"`

Objects for which `obj.__bool__()` returns `True`

Any object which does not define `__len__` or `__bool__`

Falsy

Objects for which `bool(obj)` returns `False`

Zero: `0`, `False`, `0.0`

Empty collections: `[]`, `""`, `{}`

Objects for which `obj.__bool__()` returns `False`

None

Logical and Operator

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a and b
0	0	0
0	42	0
37	0	0
19	23	23

a	b	a and b
''	[]	''
()	1	()
True	None	None
{1, 2}	'Yes'	'Yes'

If a is falsy, the result must be falsy, so return a

If a is truthy, the result depends on whether b is truthy, so return b

b only needs to be evaluated if a is truthy

Logical `or` Operator

a	b	a and b
False	False	False
False	True	True
True	False	True
True	True	True

If a is falsy, the result depends on whether b is truthy, so return b

If a is truthy, return a

b only needs to be evaluated if a is falsy

a	b	a and b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a and b
0	0	0
0	42	42
37	0	37
19	23	19

a	b	a and b
''	[]	''
()	1	1
True	None	True
{1, 2}	'Yes'	{1, 2}

Logical Operators

and

Returns an operand

Only evaluates right if left is **truthy**

or

Returns an operand

Only evaluates right if left is **falsy**

not

Always returns True or False

Always evaluates operand

Coalescing Nulls

Logical-or as a Null-coalescing Operator

```
possibly_null_value or value_if_null
```

Logical-or as a Null-coalescing Operator

```
possible_none_value or value_if_none
```

images.py

```
"""Image size functions."""


def image_width(num_pixels=None):
    return num_pixels if (num_pixels is not None) else 1280


def image_height(num_pixels=None):
    if num_pixels is None:
        num_pixels = 720
    return num_pixels


def num_image_pixels(h=None, v=None):
    return image_width(h) * image_height(v)

```


WRONG
WAY

Fallbacks Using Logical-or

```
integer_divisors or [1]

scale_factor or 1.0

text or "<empty>"
```

Guarding Expressions

Share out Laptop Stickers

guard.py

```
"""Sharing marketing loot."""

def share_stickers(num_stickers, num_developers):
    return num_developers and num_stickers // num_developers

```

num_developers (0) is falsy

Python Console

```
Python Console
>>> from guard import *
>>> share_stickers(6, 2)
3
>>> share_stickers(10, 0)
0

>>>
```

```python
def share_stickers(num_stickers, num_developers):
    if num_developers == 0:
        return 0
    return num_stickers // num_developers
```

Look before you leap:
Rule in plain sight.

```python
def share_stickers(num_stickers, num_developers):
    try:
        return num_stickers // num_developers
    except ZeroDivisionError:
        return 0
```

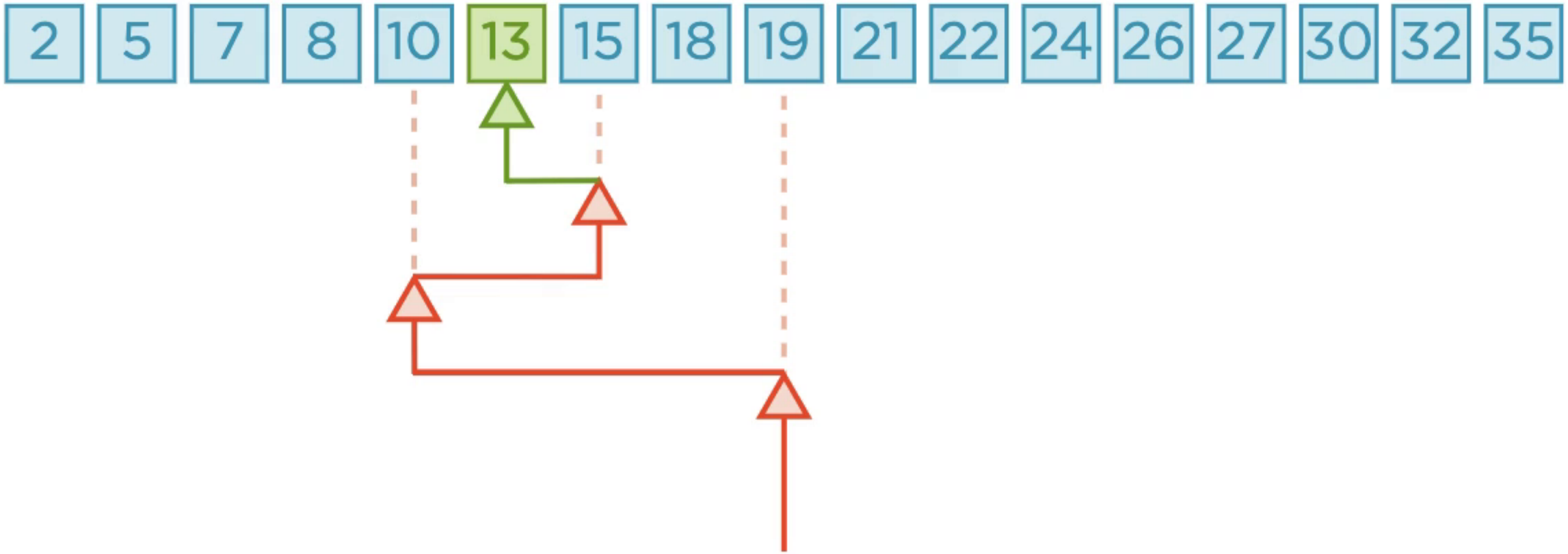
Easier to ask forgiveness than permission:
More Pythonic?

```python
def share_stickers(num_stickers, num_developers):
    return num_developers and num_stickers // num_developers
```

Shortcut-evaluation:
Concise. But obvious?

Safe Binary Search

Search by Repeated Bisection



Sorted Sequence

```
s = [5, 8, 19, 34, 35, 53]
```

Computational Complexity

Linear search: 1 000 000 comparisons for 1 000 000 elements

Binary search: 20 comparisons for 1 000 000 elements

search > search.py

search.py

```python
from bisect import bisect_left

s = [5, 8, 19, 34, 35, 53]


def contains(sequence, value):
    index = bisect_left(sequence, value)
    return (index != len(sequence)) and (sequence[index] == value)

```

Python Console

```
False
>>> contains(s, 8)
True
>>> contains(s, 12)
False
>>> contains(s, 70)
False

>>>
```

1:1 LF UTF-8 4 spaces Python 3.8 (coalescing)

Shortcut Logical Operators

Logical-or based fallbacks

Logical-and based guards

If-statements are an alternative

Conditional expressions are an alternative

Shortcut-evaluation is a common language feature

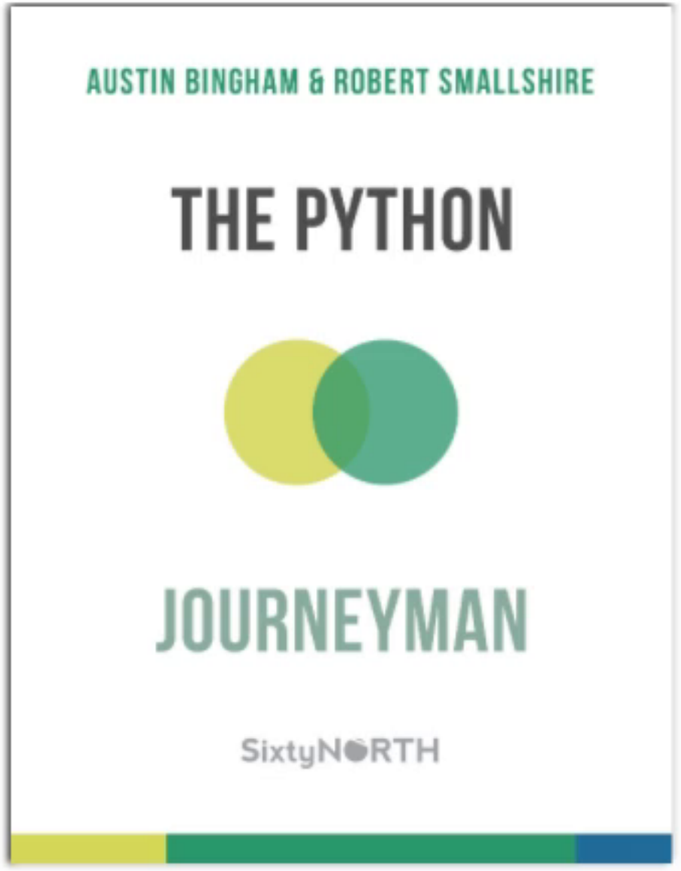
Logical operators returning operands is an unusual language feature

Core Python

on

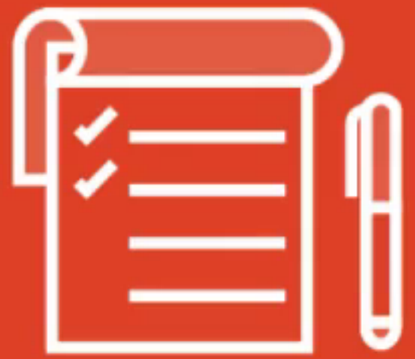
PLURALSIGHT

The Python Craftsman



leanpub.com/b/python-craftsman

Summary



While-else executed when condition becomes false

Loop-else commented "no-break"

For-else executed when iterable is exhausted

Extract loops using else blocks

Try-else for non-exceptional cases

Mappings of callables emulate switch

Overload functions using `singledispatch`

Shortcut logical operators for conditional evaluation

Well done!

Happy Programming!