Scalar Types

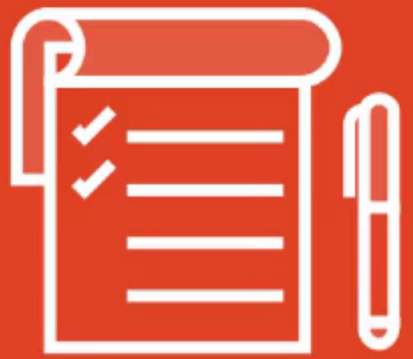
Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Overview



Python's fundamental scalar types

Basic use of relational operators

Basic flow-control mechanisms

Scalar Types

Scalar Types

arbitrary precision integer

64-bit floating point numbers

the null object

boolean logical values

`int`

unlimited precision signed integer

Int

```
>>> 10
10
>>> 0b10
2
>>> 0o10
8
>>> 0x10
16
>>> int(3.5)
3
>>> int(-3.5)
-3
>>> int("496")
496
>>> int("10000", 3)
81
>>>
```

`float`

IEEE-754 double-precision with 53-bits of binary precision

15-16 significant digits in decimal

Float

```
>>> 3.125
3.125
>>> 3e8
300000000.0
>>> 1.616e-35
1.616e-35
>>> float(7)
7.0
>>> float("1.618")
1.618
>>> float("nan")
nan
>>> float("inf")
inf
>>> float("-inf")
-inf
>>> 3.0 + 1
4.0
>>>
```

None

Null value

Often represents the absence of a value

None

```
>>> None
>>> a = None
>>> a is None
True
>>>
```

`bool`

Boolean logical values

Bool

```
False
>>> bool(42)
True
>>> bool(-1)
True
>>> bool(0.0)
False
>>> bool(0.207)
True
>>> bool(-1.117)
True
>>> bool([])
False
>>> bool([1, 5, 9])
True
>>> bool("")
False
>>> bool("Spam")
True
>>> bool("False")
True
>>> bool("True")
True
>>>
```

Relational Operators

Relational Operators

==	value equality / equivalence
!=	value inequality / inequivalence
<	less-than
>	greater-than
<=	less-than or equal
>=	greater-than or equal

Value Equality

==

True

Relational Operators

```
>>> g = 20
>>> g == 20
True
>>> g == 13
False
>>> g != 20
False
>>> g != 13
True
>>> g < 30
True
>>> g <= 20
True
>>> g > 30
False
>>> g >= 20
True
>>>
```

Control Flow

Conditional statement

Branch execution based on the value of an expression

If-statement Syntax

```
if expression:
    block
```

If-statement

```
>>> if True:
...     print("It's true!")
...
It's true!
>>> if False:
...     print("It's true!")
...
>>> if bool("eggs"):
...     print("Yes please!")
...
Yes please!
>>> if "eggs":
...     print("Yes please!")
...
Yes please!
>>>
```

Else-clause

```
>>> if h > 50:
...     print("Greater than 50")
... else:
...     print("50 or smaller")
...
50 or smaller
>>> if h > 50:
...     print("Greater than 50")
... else:
...     if h < 20:
...         print("Less than 20")
...     else:
...         print("Between 20 and 50")
...
Between 20 and 50
>>> if h > 50:
...     print("Greater than 50")
... elif h < 20:
...     print("Less than 20")
... else:
...     print("Between 20 and 50")
...
Between 20 and 50
>>>
```

While-loops

While-loops

```
while expression:
    block
```

converted to boolean

While-loops

```
>>> c = 5
>>> while c != 0:
...     print(c)
...     c -= 1
...
5
4
3
2
1
>>> c = 5
>>> while c:
...     print(c)
...     c -= 1
...
5
4
3
2
1
>>>
```

Int Truthiness

```
bool(5) == True
bool(4) == True

...
bool(0) == False
```

Explicit is better than implicit

Relational Operators

```
>>> while True:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>>
```

break

Many languages support a loop ending in a predicate test

C, C++, C#, and Java have do-while

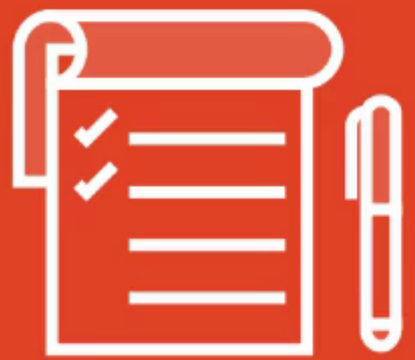
Python requires you to use `while True` and `break`

`break` jumps out of the inner-most executing loop to the line immediately after it

Break

```
>>> while True:
...     response = input()
...     if int(response) % 7 == 0:
...         break
...
12
67
34
28
>>>
```

Summary



`int`, `float`, `None`, and `bool`

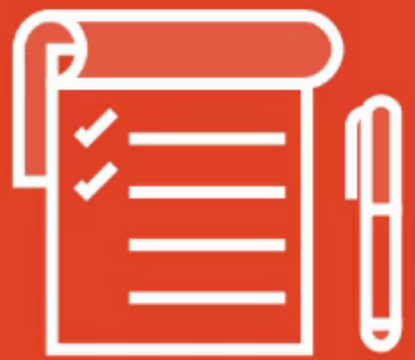
Relational operators for equivalence and ordering

Conditional code with `if-elif-else`

While-loops

While-loop expressions converted to bool

Summary



Interrupt loops with Control-C

Control-C generates a `KeyboardInterrupt` exception

Break out of loops with `break`

- Exits the inner-most executing loop
- Takes execution to the first statement following the loop

Augmented assignment operators like `+=`

Request text input from the user with `input()`