If and Comparisons 31.01.2023, 15:21

Pvthon Guide

About Python

Python Interpreter

Command Line

Keyboard Shortcuts

Style1

Style Readable

Style Décomp

Variables

Math

Functions

Debugging

Doctests

For Loop

While Loop

If and Comparisons

Boolean and or not

Range

Strings

print() Standard Out

input()

File Read Write

Lists

main() Command Line Args

Dicts

Python No Copy / is

Map

Comprehens

Sorting

If and Comparisons

The if-statement uses comparisons like i < 6 to control if lines run or not.

- If Statement
- If else:
- Comparison Operators
- If elif:
- Truthy Tests

If Statement

The if-statement controls if some lines run or not. A "boolean" is a value which is True or False. The if-statement has a boolean-test, a colon, and indented lines of code (similar to "while"):

```
if boolean-test:
  indented body lines
```

The if-statement first evaluates the the boolean test, and if it is True, runs the "body" lines once. Otherwise if the test is False, the body lines are skipped and the run continues after the last body line.

The simplest and most common sort of boolean test uses == (two equal signs next to each other) to compare two values, yielding True if the two are the same.

Here is an example that shows an if-statement inside a for-loop. Every time through the loop, the test num == 6 is evaluated, yielding boolean True or False each time.

```
>>> for num in [2, 4, 6, 8]:
                    if num == 6:
                         print('Here comes 6!')
                    print (num)
           2
Tuples
           Here comes 6!
           6
Lambda
           8
```

If test = vs. == Syntax Error

31.01.2023, 15:21 If and Comparisons

Pvthon Guide

About Python It's very easy to accidentally type a single equal sign for a comparison like the following, but in Python that is flagged as a syntax error:

```
Python
Interpreter
```

if num = 6: # typo, meant == print('hi')

Command Line

Style: Do Not Write x == True

Keyboard Shortcuts

Suppose some foo() function is supposed to return True or False. Do not write an if-test like this:

```
Style1
```

Style Readable

if foo() == True: # NO, do not use == True print('yay')

Style Décomp

Instead, let the if/while take the boolean value directly like this:

```
Variables
Math
```

```
if foo():
                     # YES this way
    print('yay')
```

Functions Debugging

Doctests For Loop

Or to test if foo() is False use not: if not foo(): print('yay is cancelled')

While Loop

If and Comparisons

if else:

Boolean and

or not

Range

Strings

print() Standard

Out

input()

File Read Write

Lists

Dicts

main() Command Line Args

1 or action-2 depending on the test, but not do nothing. Use regular if to do action-1 or nothing.

The optional else: part of an if-statement adds code to run in the case that

the test is False. Use else: if the run should choose between either action-

```
# set message according to score
if score > high score:
    message = 'New high score!'
else:
    message = 'Oh well!
```

To run code if a test is False and otherwise do nothing, use not like this:

```
if not foo():
    message = 'no foo today'
```

Copy / is **Tuples**

Python No

Boolean Comparison Operators

Map Lambda The most common way to get a boolean True/False is comparing two values, e.g. the comparison expression num == 6 evaluates to True when num is 6 and False otherwise.

Comprehens

Sorting

Comparison operators:

Python Guide

About Python == test if two values are equal (2 equals signs together). Works for int, string, list, dict, .. most types. Not recommended to use with float values.

Python Interpreter

! = not-equal, the opposite of equal (uses == under the hood)

Command Line

Keyboard

Shortcuts

< = less-than, less-or-equal. These work for most types that have an</p> ordering: numbers, strings, dates. For strings, < provides alphabetic ordering. Uppercase letters are ordered before lowercase. It is generally an error to compare different types for less-than, e.g. this is an error: 4 <

Style Readable

Style1

Style Décomp

Variables >= greater than, greater-or-equal.

'hello'

Math

Functions The interpreter examples below shows various == style comparisons and Debugging their boolean results:

Doctests

```
>>> 5 == 6
For Loop
             False
While Loop
             >>> 5 == 5
If and
             True
Comparisons
             >>> 5 != 6
Boolean and
             True
or not
             >>> 4 > 2
Range
             True
             >>> 4 > 5
Strings
             False
print()
             >>> 4 > 4
Standard
Out
             False
             >>> 4 >= 4
                            # contrast >= vs. >
input()
             True
File Read
             >>> s = 'he' + 'llo'
Write
             >>> s == 'hello'
Lists
             True
main()
             >>> 'apple' < 'banana'
Command
Line Args
             True
             >>> 'apple' < 4
Dicts
             TypeError: '<' not supported between instances of
Python No
```

Copy / is **Tuples**

if elif:

Map Lambda

There is a more rarely used elif for where a series of if-tests can be strung together (mnemonic: 'elif' is length 4, like 'else'):

Comprehens Sorting

'str' and 'int'

Python Interpreter Command

The tests are run top to bottom, running the code for the first that is True. However, the logic of when each case runs can be hard to see. What must be true for case c below? You really have to think about the code work work out

Keyboard Shortcuts

when (c) happens.

Style1

Line

```
Style if score > high and s != 'alice':
Readable # a

Style elif s == 'bob':
Decomp # b

Variables # c
```

Functions

Answer: c happens when s is not 'bob' but also (score \leq high or s == 'alice or both)

Debugging Doctests

For Loop

While Loop

If and Comparisons

If/else chains are fine, just don't think they are trivial. Only add else if the code needs it. Nice, simple if handles most problems and is the most readable.

Boolean and

or not

Range Strings

print() Standard

Out input()

Return vs. if/else chains

Nick style idea: only use else when it is truly needed. If the code can work using only a plain-if, I prefer to write it that way. This works extra well with decomposed out functions, where 'return' can be used to bail out for the first few cases, leaving the main case below without indentation, like this:

```
File Read Write def foo(s):

Unite if len(s) < 4:

The return 'too small!'

The main() command line Args return 'too big!'

Dicts

Python No computation having screened out too-short and too-long short and too-long short and too-long stores.
```

Copy/is short and too-long

Tuples # notice: no if-else structure, not indented down here

Map

Comprehens

Sorting

Lambda

You can use else if you prefer, just thinking about possible alternative structure here.

. . .

31.01.2023, 15:21 If and Comparisons

Python Guide

Truthy True/False

About Python We think of if/while tests as looking at boolean values, however the rules are flexible so any type can work in there.

Python Interpreter

What does this code do?

Command Line

Keyboard

Shortcuts

```
s = 'hello'
if s:
    print('truthy')
```

Style1

Style Readable What does s mean as an if-test? Python has rules for this we'll call "truthy" logic.

Style Decomp

Variables Truthy Logic

Math

The truthy rules define a series of values which count as False. All the sort of "empty" values count as False: 0, 0.0, None, '', [], {}

Debugging

Functions

When an if-test expression is something like 0 or None or the empty string

For Loop '', that counts as False. Any other values counts as True. The int 6 or the

non-empty string 'hi', or the list [1, 2] all count as True.

If and Comparisons

While Loop

The bool () function takes any value and returns a formal bool

Boolean and or not

False/True value, so it's a way for us to see how truthy logic works in the

non-zero int - True

Range interpreter:

Strings

```
>>> bool(0)
print()
            False
Standard
Out
            >>> bool(0.0)
input()
            False
            >>> bool('')
                                 empty string - False
File Read
Write
            False
            >>> bool([])
                               # empty list - False
Lists
            False
main()
            >>> bool(None)
```

Command >>> D

Line Args False

Dicts >>> bool (6)

Tuples True

>>> bool([1, 2]) # list of something - True

Map Lambda True

>>> bool('False') # tricky: what's this one?

Comprehens ??

Sorting

Truthy Shorthand

31.01.2023, 15:21 If and Comparisons

Python Guide

About Python

Python Interpreter Why does the truthy system exist? It makes it easy to test, for example, for an empty string like the following. Testing for "empty" data is such a common case, it's nice to have a shorthand for it. For CS106A, you don't ever need to use this shorthand, but it's there if you want to use it.

```
Command Line
```

Keyboard Shortcuts

Style1

Style Readable

Style Decomp

Variables

Math

Functions

Debugging

Doctests

For Loop

While Loop

If and Comparisons

Boolean and or not

Range

Strings

print() Standard Out

input()

File Read Write

Lists

main() Command Line Args

Dicts

Python No Copy / is

Tuples

Map Lambda

Comprehens

Sorting

```
# long form screen out empty string
if len(s) > 0:
    print(s)

# shorter way, handy!
if s:
    print(s)
```

Copyright 2020 Nick Parlante