Asking the user for input until they give a valid response

170

73

I am writing a program that must accept input from the user.

```
#note: Python 2.7 users should use `raw_input`, the equivalent of 3.X's `input`
age = int(input("Please enter your age: "))
if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

This works as expected if the user enters sensible data.

```
C:\Python\Projects> canyouvote.py
Please enter your age: 23
You are able to vote in the United States!
```

But if they make a mistake, then it crashes:

```
C:\Python\Projects> canyouvote.py
Please enter your age: dickety six
Traceback (most recent call last):
  File "canyouvote.py", line 1, in <module>
    age = int(input("Please enter your age: "))
ValueError: invalid literal for int() with base 10: 'dickety six'
```

Instead of crashing, I would like it to try getting the input again. Like this:

```
C:\Python\Projects> canyouvote.py
Please enter your age: dickety six
Sorry, I didn't understand that.
Please enter your age: 26
You are able to vote in the United States!
```

How can I accomplish this? What if I also wanted to reject values like `-1`, which is a valid `int`, but nonsensical in this context?

python | validation | loops | python-3.x | user-input

share edit

edited Oct 16 '14 at 6:27

community wiki
8 revs, 5 users 96%
Kevin

18 PS. Some may think it's wrong that I'm answering my own question right after posting it. Before downvoting, please read It's OK to Ask and Answer Your Own Questions. See also Putting the Community back in Wiki, which says "Compiling a canonical reference" is "something wonderful". Questions like this one are asked often enough to justify writing a post that can concisely answer all of them. – Kevin Apr 25 '14 at 13:32

add a comment

asked 2 years ago
viewed 78328 times
active 1 month ago

FEATURED ON META

Documentation Update, August 4th

HOT META POSTS

56 Why are questions on the algorithms tag downvoted/closed so aggressively?

23 What level of experience should we assume for readers of SO Documentation?

12 Can we delete "Setting up a git repo on my GoDaddy hosting plan"?

8 Answers

active | oldest | votes

235

+400

The simplest way to accomplish this would be to put the `input` method in a while loop. Use `continue` when you get bad input, and `break` out of the loop when you're satisfied.

When Your Input Might Raise an Exception

Use try and catch to detect when the user enters data that can't be parsed.

```
while True:
    try:
        # Note: Python 2.x users should use raw_input, the equivalent of 3.x's input
        age = int(input("Please enter your age: "))
    except ValueError:
        print("Sorry, I didn't understand that.")
        #better try again... Return to the start of the loop
        continue
    else:
        #age was successfully parsed!
        #we're ready to exit the loop.
        break
if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

Implementing Your Own Validation Rules

If you want to reject values that Python can successfully parse, you can add your own validation logic.

```
while True:
    data = input("Please enter a loud message (must be all caps): ")
    if not data.isupper():
        print("Sorry, your response was not loud enough.")
        continue
    else:
        #we're happy with the value given.
        #we're ready to exit the loop.
        break

while True:
    data = input("Pick an answer from A to D:")
    if data.lower() not in ('a', 'b', 'c', 'd'):
        print("Not an appropriate choice.")
    else:
        break
```

Combining Exception Handling and Custom Validation

Linked

2 Loop until a specific user input

0 python repeat program while true

-2 Check if input is positive integer

1 Go back to start or repeat raw input a number of times

0 How to repeat function in Python depending on output?

3 Most Pythonic way to do input validation

0 Ask user to input a correct response in Python?

-3 Loop on if-statement to reject invalid input

1 How do I check a value entered?? Python 3.4

-1 How to check if input is text

see more linked questions...

Related

734 What's the best method for sanitizing user input with PHP?

0 Hangman user input validation

0 Write a program in python that in a loop asks the user for a integer until the user

Both of the above techniques can be combined into one loop.

```
while True:
    try:
        age = int(input("Please enter your age: "))
    except ValueError:
        print("Sorry, I didn't understand that.")
        continue

    if age < 0:
        print("Sorry, your response must not be negative.")
        continue
    else:
        #age was successfully parsed, and we're happy with its value.
        #we're ready to exit the loop.
        break
if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

Encapsulating it All in a Function

If you need to ask your user for a lot of different values, it might be useful to put this code in a function, so you don't have to retype it every time.

```
def get_non_negative_int(prompt):
    while True:
        try:
            value = int(input(prompt))
        except ValueError:
            print("Sorry, I didn't understand that.")
            continue

        if value < 0:
            print("Sorry, your response must not be negative.")
            continue
        else:
            break
    return value

age = get_non_negative_int("Please enter your age: ")
kids = get_non_negative_int("Please enter the number of children you have: ")
salary = get_non_negative_int("Please enter your yearly earnings, in dollars: ")
```

Putting it all together

You can extend this idea to make a very generic input function:

```
def sanitised_input(prompt, type_=None, min_=None, max_=None, range_=None):
    if min_ is not None and max_ is not None and max_ < min_:
        raise ValueError("min_ must be less than or equal to max_.")
    while True:
        ui = input(prompt)
        if type_ is not None:
            try:
                ui = type_(ui)
            except ValueError:
                print("Input type must be {0}.".format(type_.__name__))
                continue
        if max_ is not None and ui > max_:
            print("Input must be less than or equal to {0}.".format(max_))
        elif min_ is not None and ui < min_:
            print("Input must be greater than or equal to {0}.".format(min_))
        elif range_ is not None and ui not in range_:
            if isinstance(range_, range):
                template = "Input must be between {0.start} and {0.stop}."
                print(template.format(range_))
            else:
                template = "Input must be {0}."
                if len(range_) == 1:
                    print(template.format(*range_))
                else:
                    print(template.format(" or ".join((", ".join(map(str,
                                                                     range_[:-1])),
                                                       str(range_[-1])))))
        else:
            return ui
```

With usage such as:

```
age = sanitised_input("Enter your age: ", int, 1, 101)
answer = sanitised_input("Enter your answer", str.lower, range_=('a', 'b', 'c', 'd'))
```

Common Pitfalls, and Why you Should Avoid Them

The Redundant Use of Redundant `input` Statements

This method works but is generally considered poor style:

```
data = input("Please enter a loud message (must be all caps): ")
while not data.isupper():
    print("Sorry, your response was not loud enough.")
    data = input("Please enter a loud message (must be all caps): ")
```

It might look attractive initially because it's shorter than the `while True` method, but it violates the Don't Repeat Yourself principle of software development. This increases the likelihood of bugs in your system. What if you want to backport to 2.7 by changing `input` to `raw_input`, but accidentally change only the first `input` above? It's a `SyntaxError` just waiting to happen.

Recursion Will Blow Your Stack

If you've just learned about recursion, you might be tempted to use it in `get_non_negative_int` so you can dispose of the while loop.

```
def get_non_negative_int(prompt):
    try:
        value = int(input(prompt))
    except ValueError:
        print("Sorry, I didn't understand that.")
        return get_non_negative_int(prompt)

    if value < 0:
        print("Sorry, your response must not be negative.")
        return get_non_negative_int(prompt)
    else:
        return value
```

This appears to work fine most of the time, but if the user enters invalid data enough times, the script will terminate with a `RuntimeError: maximum recursion depth exceeded`. You may think "no fool would make 1000 mistakes in a row", but you're underestimating the ingenuity of fools!

share edit

edited Dec 8 '14 at 12:01

community wiki
4 revs, 2 users 83%
Kevin

add a comment

10

Why would you do a `while True` and then break out of this loop while you can also just put your requirements in the while statement since all you want is to stop once you have the age?

```
age = None
while age is None:
    input_value = raw_input("Please enter your age: ")
    try:
        # try and convert the string input to a number
        age = int(input_value)
    except ValueError:
        # tell the user off
        print "{input} is not a number, please enter a number only".format(input=input_value)
if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

This would result in the following:

```
Please enter your age: *potato*
potato is not a number, please enter a number only
Please enter your age: *5*
You are not able to vote in the United States.
```

this will work since age will never have a value that will not make sense and the code follows the logic of your "business process"

share edit

answered Jan 14 at 12:43

community wiki
Steven Stip

add a comment

9

Though the accepted answer is amazing. I would also like to share a quick hack for this problem. (This takes care of the negative age problem as well.)

```
f=lambda age: (age.isdigit() and ((int(age)>=18  and "Can vote" ) or "Cannot vote")) or \
f(raw_input("invalid input. Try again\nPlease enter your age: "))
print f(raw_input("Please enter your age: "))
```

P.S. This code is for python 2.x and can be exported to 3.x by changing the raw_input and print functions.

share edit

answered Jun 28 '15 at 23:29

community wiki
aaveg

Note that this code is recursive, but recursion isn't necessary here, and as Kevin said, it can blow your stack. – PM 2Ring Jan 31 at 8:12

@PM2Ring - you are right. But my purpose here was just to show how "short circuiting" can minimise (beautify) long pieces of code. – aaveg Feb 3 at 8:58

add a comment

3

So, I was messing around with something similar to this recently, and I came up with the following solution, which uses a way of getting input that rejects junk, before it's even checked in any logical way.

`read_single_keypress()` courtesy http://stackoverflow.com/a/6599441/4532996

```
def read_single_keypress() -> str:
    """Waits for a single keypress on stdin.
    -- from :: http://stackoverflow.com/a/6599441/4532996
    """

    import termios, fcntl, sys, os
    fd = sys.stdin.fileno()
    # save old state
    flags_save = fcntl.fcntl(fd, fcntl.F_GETFL)
    attrs_save = termios.tcgetattr(fd)
    # make raw - the way to do this comes from the termios(3) man page.
    attrs = list(attrs_save) # copy the stored version to update
    # iflag
    attrs[0] &= ~(termios.IGNBRK | termios.BRKINT | termios.PARMRK
                  | termios.ISTRIP | termios.INLCR | termios. IGNCR
                  | termios.ICRNL | termios.IXON )
    # oflag
    attrs[1] &= ~termios.OPOST
    # cflag
    attrs[2] &= ~(termios.CSIZE | termios. PARENB)
    attrs[2] |= termios.CS8
    # lflag
    attrs[3] &= ~(termios.ECHONL | termios.ECHO | termios.ICANON
                  | termios.ISIG | termios.IEXTEN)
    termios.tcsetattr(fd, termios.TCSANOW, attrs)
    # turn off non-blocking
    fcntl.fcntl(fd, fcntl.F_SETFL, flags_save & ~os.O_NONBLOCK)
    # read a single keystroke
    try:
        ret = sys.stdin.read(1) # returns a single character
    except KeyboardInterrupt:
        ret = 0
    finally:
        # restore old state
        termios.tcsetattr(fd, termios.TCSAFLUSH, attrs_save)
        fcntl.fcntl(fd, fcntl.F_SETFL, flags_save)
```

You can find the complete module here.

Example:

```
$ ./input_constrain.py
can you vote? age : a
sorry, age can only consist of digits.
$ ./input_constrain.py
can you vote? age : 23<RETURN>
your age is 23
You can vote!
$ _
```

Note that the nature of this implementation is it closes stdin as soon as something that isn't a digit is read. I didn't hit enter after `a`, but I needed to after the numbers.

You could merge this with the `thismany()` function in the same module to only allow, say, three digits.

share edit

edited Jan 31 at 3:52

community wiki
2 revs
cat

add a comment

1

```
def validate_age(age):
    if age >=0 :
        return True
    return False

while True:
    try:
        age = int(raw_input("Please enter your age:"))
        if validate_age(age): break
    except ValueError:
        print "Error: Invalid age."
```

share edit

answered Jun 23 at 10:34

community wiki
ojas mohril

add a comment

1

While a `try` / `except` block will work, a much faster and cleaner way to accomplish this task would be to use `str.isdigit()`.

```
while True:
    age = input("Please enter your age: ")
    if age.isdigit():
        age = int(age)
        break
    else:
        print("Invalid number '{age}'. Try again.".format(age=age))

if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

share edit

edited Jun 6 at 7:15

community wiki
2 revs
2Cubed

`str.isnumeric` is only available in Python 3 and does not return true for all valid integers. Like `str.isdigit` it is testing properties of the characters, and `-` is not a numeric character. – Martijn Pieters ♦ Jun 6 at 6:50

add a comment

-3

If you want a vaild, no number, response, I would do this:

```
age = input("Please enter your age: ")
while age == '':
    print("Sorry, I didn't understand that")
    age = input("Please enter your age: ")
while not age.isalpha():
    if age >= 18:
        print("You are able to vote in the United States!")
        break
    elif age < 18:
        print("You are not able to vote in the United States.")
        break
    else:
        print("Sorry, I didn't understand that")
        age = int(input("Please enter your age: "))
    while age == type(float):
        print("Sorry, I didn't understand that")
        age = input("Please enter your age: ")

else:
    print("Sorry, I didn't understand that")
    age = input("Please enter your age: ")
```

This makes sure that the type of input isn't a float, and that it is a number. If they just press `enter` it will ask their input again.

share edit

edited Oct 2 '15 at 17:57

community wiki
2 revs, 2 users 93%
Slass33

This repeats the input prompt, and does the wrong thing if the user does not provide an empty input the first time. – tripleee Jan 9 at 18:46

add a comment

-4

```
import re

pattern = r'^[-+]?[0-9]*$'
while True:
    string = input("age >> ")
    if re.match(pattern, string) is not None:
        age = int(string)
        break

if age >= 18:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

share edit

answered Nov 17 '15 at 18:18

community wiki
Sergey Nosov

add a comment

protected by Robert Harvey ♦ Jan 14 '15 at 21:13

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?

Not the answer you're looking for? Browse other questions tagged python validation loops python-3.x user-input or ask your own question.

question feed