Asking the user for input until they give a valid response

Asked 10 years, 8 months ago Modified 22 days ago Viewed 1.1m times

764

I am writing a program that accepts user input.

```
#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.")
```

The program works as expected as long as the the user enters meaningful data.

```
Please enter your age: 23
You are able to vote in the United States!
```

But it fails if the user enters invalid data:

```
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 the program to ask for the input again. Like this:

```
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 do I ask for valid input instead of crashing or accepting invalid values (e.g. `-1`)?

python validation input

edited May 22, 2022 at 19:18

community wiki
14 revs, 10 users 55%
Kevin

1002

The simplest way to accomplish this is 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 `except` 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

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:
                    expected = " or ".join((
                        ", ".join(str(x) for x in range_[:-1]),
                        str(range_[-1])
                    ))
                    print(template.format(expected))
        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

Improve this answer

Follow

edited Jun 6, 2020 at 17:56

community wiki
9 revs, 7 users 73%
Kevin

2 Don't estimate the ingenuity of fools... and clever attackers. A DOS attack would be easiest for this sort of thing, but others may be possible. – Solomon Ucko Apr 28, 2019 at 2:53

Can we use the new "walrus" operator instead of redundant inputs? Is it also a poor style? – J Arun Mani Apr 16, 2020 at 7:52

1 @JArunMani I don't think it would be poor style, but might be a little less readable. You will indeed have only one `input` per loop and the loop will become very short, but the condition might become pretty long... – Tomerikoo May 9, 2020 at 8:40

could you specify the license of this to be public domain or MIT (or even specify in your profile), so noone has to worry about the cc license which does not allow for use in non-cc works? – laundmo Jul 23, 2020 at 23:34

6 @laundmo,certainly I release the code blocks that I wrote into the public domain. Feel free to use them in any context, without my explicit permission or knowledge. Regarding the non-code-block segments, If you want to paste my entire answer into a "Learn Python" book you're writing, let's talk royalties ;-) – Kevin Jul 24, 2020 at 13:50

65

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 = 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

Improve this answer

Follow

edited Nov 28 at 12:25

community wiki
3 revs, 3 users 86%
Steven Stip

A well-designed **exit-condition** 🤩 like advised here avoids the pitfall of infinite loop caused by `while True` without safely reaching break or return. – hc_dev Jun 17, 2022 at 17:48

43

Functional approach or "*look mum no loops!*":

```
from itertools import chain, repeat

prompts = chain(["Enter a number: "], repeat("Not a number! Try again: "))
replies = map(input, prompts)
valid_response = next(filter(str.isdigit, replies))
print(valid_response)
```

```
Enter a number:  a
Not a number! Try again:  b
```

```
Not a number! Try again:  1
1
```

or if you want to have a "bad input" message separated from an input prompt as in other answers:

```
prompt_msg = "Enter a number: "
bad_input_msg = "Sorry, I didn't understand that."
prompts = chain([prompt_msg], repeat('\n'.join([bad_input_msg, prompt_msg])))
replies = map(input, prompts)
valid_response = next(filter(str.isdigit, replies))
print(valid_response)
```

```
Enter a number:  a
Sorry, I didn't understand that.
Enter a number:  b
Sorry, I didn't understand that.
Enter a number:  1
1
```

How does it work?

1. `prompts = chain(["Enter a number: "], repeat("Not a number! Try again: "))`

   This combination of `itertools.chain` and `itertools.repeat` will create an iterator which will yield strings `"Enter a number: "` once, and `"Not a number! Try again: "` an infinite number of times:

   ```
   for prompt in prompts:
       print(prompt)
   ```

   ```
   Enter a number:
   Not a number! Try again:
   Not a number! Try again:
   Not a number! Try again:
   # ... and so on
   ```

2. `replies = map(input, prompts)` - here `map` will apply all the `prompts` strings from the previous step to the `input` function. E.g.:

   ```
   for reply in replies:
       print(reply)
   ```

   ```
   Enter a number:  a
   a
   Not a number! Try again:  1
   1
   Not a number! Try again:  it doesn't care now
   ```

```
it doesn't care now
# and so on...
```

3. We use `filter` and `str.isdigit` to filter out those strings that contain only digits:

```
only_digits = filter(str.isdigit, replies)
for reply in only_digits:
    print(reply)
```

```
Enter a number:  a
Not a number! Try again:  1
1
Not a number! Try again:  2
2
Not a number! Try again:  b
Not a number! Try again: # and so on...
```

And to get only the first digits-only string we use `next`.

Other validation rules:

1. **String methods:** Of course you can use other string methods like `str.isalpha` to get only alphabetic strings, or `str.isupper` to get only uppercase. See docs for the full list.
2. **Membership testing:**
   There are several different ways to perform it. One of them is by using `__contains__` method:

```
from itertools import chain, repeat

fruits = {'apple', 'orange', 'peach'}
prompts = chain(["Enter a fruit: "], repeat("I don't know this one! Try agai
"))
replies = map(input, prompts)
valid_response = next(filter(fruits.__contains__, replies))
print(valid_response)
```

```
Enter a fruit:  1
I don't know this one! Try again:  foo
I don't know this one! Try again:  apple
apple
```

3. **Numbers comparison:**
   There are useful comparison methods which we can use here. For example, for `__lt__` (`<`):

```
from itertools import chain, repeat

prompts = chain(["Enter a positive number:"], repeat("I need a positive numb
Try again:"))
```

```
replies = map(input, prompts)
numeric_strings = filter(str.isnumeric, replies)
numbers = map(float, numeric_strings)
is_positive = (0.).__lt__
valid_response = next(filter(is_positive, numbers))
print(valid_response)
```

```
Enter a positive number: a
I need a positive number! Try again: -5
I need a positive number! Try again: 0
I need a positive number! Try again: 5
5.0
```

Or, if you don't like using dunder methods (dunder = double-underscore), you can always define your own function, or use the ones from the `operator` module.

4. **Path existance:**

   Here one can use `pathlib` library and its `Path.exists` method:

```
from itertools import chain, repeat
from pathlib import Path

prompts = chain(["Enter a path: "], repeat("This path doesn't exist! Try aga
"))
replies = map(input, prompts)
paths = map(Path, replies)
valid_response = next(filter(Path.exists, paths))
print(valid_response)
```

```
Enter a path:  a b c
This path doesn't exist! Try again:  1
This path doesn't exist! Try again:  existing_file.txt
existing_file.txt
```

Limiting number of tries:

If you don't want to torture a user by asking him something an infinite number of times, you can specify a limit in a call of `itertools.repeat`. This can be combined with providing a default value to the `next` function:

```
from itertools import chain, repeat

prompts = chain(["Enter a number:"], repeat("Not a number! Try again:", 2))
replies = map(input, prompts)
valid_response = next(filter(str.isdigit, replies), None)
print("You've failed miserably!" if valid_response is None else 'Well done!')
```

```
Enter a number: a
Not a number! Try again: b
```

```
Not a number! Try again: c
You've failed miserably!
```

Preprocessing input data:

Sometimes we don't want to reject an input if the user accidentally supplied it *IN CAPS* or with a space in the beginning or an end of the string. To take these simple mistakes into account we can preprocess the input data by applying `str.lower` and `str.strip` methods. For example, for the case of membership testing the code will look like this:

```
from itertools import chain, repeat

fruits = {'apple', 'orange', 'peach'}
prompts = chain(["Enter a fruit: "], repeat("I don't know this one! Try again:
"))
replies = map(input, prompts)
lowercased_replies = map(str.lower, replies)
stripped_replies = map(str.strip, lowercased_replies)
valid_response = next(filter(fruits.__contains__, stripped_replies))
print(valid_response)
```

```
Enter a fruit:  duck
I don't know this one! Try again:     Orange
orange
```

In the case when you have many functions to use for preprocessing, it might be easier to use a function performing a function composition. For example, using the one from here:

```
from itertools import chain, repeat

from lz.functional import compose

fruits = {'apple', 'orange', 'peach'}
prompts = chain(["Enter a fruit: "], repeat("I don't know this one! Try again:
"))
replies = map(input, prompts)
process = compose(str.strip, str.lower)  # you can add more functions here
processed_replies = map(process, replies)
valid_response = next(filter(fruits.__contains__, processed_replies))
print(valid_response)
```

```
Enter a fruit:  potato
I don't know this one! Try again:   PEACH
peach
```

Combining validation rules:

For a simple case, for example, when the program asks for age between 1 and 120, one can just add another `filter`:

```
from itertools import chain, repeat

prompt_msg = "Enter your age (1-120): "
bad_input_msg = "Wrong input."
prompts = chain([prompt_msg], repeat('\n'.join([bad_input_msg, prompt_msg])))
replies = map(input, prompts)
numeric_replies = filter(str.isdigit, replies)
ages = map(int, numeric_replies)
positive_ages = filter((0).__lt__, ages)
not_too_big_ages = filter((120).__ge__, positive_ages)
valid_response = next(not_too_big_ages)
print(valid_response)
```

But in the case when there are many rules, it's better to implement a function performing a logical conjunction. In the following example I will use a ready one from here:

```
from functools import partial
from itertools import chain, repeat

from lz.logical import conjoin


def is_one_letter(string: str) -> bool:
    return len(string) == 1


rules = [str.isalpha, str.isupper, is_one_letter, 'C'.__le__, 'P'.__ge__]

prompt_msg = "Enter a letter (C-P): "
bad_input_msg = "Wrong input."
prompts = chain([prompt_msg], repeat('\n'.join([bad_input_msg, prompt_msg])))
replies = map(input, prompts)
valid_response = next(filter(conjoin(*rules), replies))
print(valid_response)
```

```
Enter a letter (C-P):  5
Wrong input.
Enter a letter (C-P):  f
Wrong input.
Enter a letter (C-P):  CDE
Wrong input.
Enter a letter (C-P):  Q
Wrong input.
Enter a letter (C-P):  N
N
```

Unfortunately, if someone needs a custom message for each failed case, then, I'm afraid, there is no *pretty* functional way. Or, at least, I couldn't find one.

Share

Improve this answer

Follow

edited Jun 20, 2020 at 9:12

community wiki
3 revs
Georgy

30

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(input("invalid input. Try again\nPlease enter your age: "))
print(f(input("Please enter your age: ")))
```

P.S. This code is for python 3.x.

Share

Improve this answer

Follow

edited Dec 21, 2018 at 18:25

community wiki
2 revs, 2 users 78%
aaveg

1 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, 2016 at 8:12

3 @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, 2016 at 8:58

15 Why would you assign a lambda to a variable, just use `def` instead. `def f(age):` is far clearer than `f = lambda age:` – GP89 May 16, 2017 at 22:29

4 In some cases, you may need the age just once and then there is no use of that function. One may want to use a function and throw it away after the job is done. Also, this may not be the best way, but it definitely is a different way of doing it (which was the purpose of my solution). – aaveg May 16, 2017 at 23:17

@aaveg how would you turn this code to actually save the age provided by the user? – Tytire Recubans Jul 4, 2019 at 20:04

23

Using *Click*:

Click is a library for command-line interfaces and it provides functionality for asking a valid response from a user.

Simple example:

```
import click

number = click.prompt('Please enter a number', type=float)
print(number)
```

```
Please enter a number:
 a
Error: a is not a valid floating point value
Please enter a number:
 10
10.0
```

Note how it converted the string value to a float automatically.

Checking if a value is within a range:

There are different custom types provided. To get a number in a specific range we can use `IntRange`:

```
age = click.prompt("What's your age?", type=click.IntRange(1, 120))
print(age)
```

```
What's your age?:
 a
Error: a is not a valid integer
What's your age?:
 0
Error: 0 is not in the valid range of 1 to 120.
What's your age?:
 5
5
```

We can also specify just one of the limits, `min` or `max`:

```
age = click.prompt("What's your age?", type=click.IntRange(min=14))
print(age)
```

```
What's your age?:
 0
Error: 0 is smaller than the minimum valid value 14.
What's your age?:
 18
18
```

Membership testing:

Using `click.Choice` type. By default this check is case-sensitive.

```
choices = {'apple', 'orange', 'peach'}
choice = click.prompt('Provide a fruit', type=click.Choice(choices,
case_sensitive=False))
print(choice)
```

```
Provide a fruit (apple, peach, orange):
 banana
Error: invalid choice: banana. (choose from apple, peach, orange)
Provide a fruit (apple, peach, orange):
 OrAnGe
orange
```

Working with paths and files:

Using a `click.Path` type we can check for existing paths and also resolve them:

```
path = click.prompt('Provide path', type=click.Path(exists=True,
resolve_path=True))
print(path)
```

```
Provide path:
 nonexistent
Error: Path "nonexistent" does not exist.
Provide path:
 existing_folder
'/path/to/existing_folder
```

Reading and writing files can be done by `click.File`:

```
file = click.prompt('In which file to write data?', type=click.File('w'))
with file.open():
    file.write('Hello!')
# More info about `lazy=True` at:
# https://click.palletsprojects.com/en/7.x/arguments/#file-opening-safety
file = click.prompt('Which file you wanna read?', type=click.File(lazy=True))
with file.open():
    print(file.read())
```

```
In which file to write data?:
         # <-- provided an empty string, which is an illegal name for a file
In which file to write data?:
 some_file.txt
Which file you wanna read?:
 nonexistent.txt
Error: Could not open file: nonexistent.txt: No such file or directory
Which file you wanna read?:
```

```
 some_file.txt
Hello!
```

Other examples:

Password confirmation:

```
password = click.prompt('Enter password', hide_input=True,
confirmation_prompt=True)
print(password)
```

```
Enter password:
 ......
Repeat for confirmation:
 .
Error: the two entered values do not match
Enter password:
 ......
Repeat for confirmation:
 ......
qwerty
```

Default values:

In this case, simply pressing Enter (or whatever key you use) without entering a value, will give you a default one:

```
number = click.prompt('Please enter a number', type=int, default=42)
print(number)
```

```
Please enter a number [42]:
 a
Error: a is not a valid integer
Please enter a number [42]:

42
```

Share

Improve this answer

Follow

edited Jun 20, 2020 at 9:12

community wiki
2 revs
Georgy

I really like click library. It also allows defining custom type based on your application needs as shown here – chetal Feb 3 at 23:15

22

I am a big fan of Unix philosophy "Do one thing and do it well". Capturing user input and validating it are two separate steps:

- prompting the user for input with `get_input` until the input is ok
- validating using a `validator` function that can be passed to `get_input`

It can be kept as simple as (Python 3.8+, with the walrus operator):

```
def get_input(
    prompt="Enter a value: ",
    validator=lambda x: True,
    error_message="Invalid input. Please try again.",
):
    while not validator(value := input(prompt)):
        print(error_message)
    return value

def is_positive_int(value):
    try:
        return int(value) >= 0
    except ValueError:
        return False

if __name__ == "__main__":
    val = get_input("Give a positive number: ", is_positive_int)
    print(f"OK, thanks for {val}")
```

Sample run:

```
Give a positive number: -5
Invalid input. Please try again.
Give a positive number: asdf
Invalid input. Please try again.
Give a positive number:
Invalid input. Please try again.
Give a positive number: 42
OK, thanks for 42
```

In Python < 3.8 you could use `get_input` like this:

```
def get_input(
    prompt="Enter a value: ",
    validator=lambda x: True,
    error_message="Invalid input. Please try again.",
):
    while True:
        value = input(prompt)
        if validator(value):
            return value
        print(error_message)
```

You might also handle `KeyboardInterrupt` and print a friendly exit message before terminating the application. A counter can be used to limit the allowed retries if desired.

Share

Improve this answer

Follow

edited Jan 28, 2022 at 6:18

community wiki
3 revs, 2 users 73%
np8

16

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 https://stackoverflow.com/a/6599441/4532996

```
def read_single_keypress() -> str:
    """Waits for a single keypress on stdin.
    -- from :: https://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)
    return ret

def until_not_multi(chars) -> str:
    """read stdin until !(chars)"""
    import sys
```

```
    chars = list(chars)
    y = ""
    sys.stdout.flush()
    while True:
        i = read_single_keypress()
        _ = sys.stdout.write(i)
        sys.stdout.flush()
        if i not in chars:
            break
        y += i
    return y

def _can_you_vote() -> str:
    """a practical example:
    test if a user can vote based purely on keypresses"""
    print("can you vote? age : ", end="")
    x = int("0" + until_not_multi("0123456789"))
    if not x:
        print("\nsorry, age can only consist of digits.")
        return
    print("your age is", x, "\nYou can vote!" if x >= 18 else "Sorry! you can't
vote")

_can_you_vote()
```

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

Improve this answer

Follow

edited May 23, 2017 at 12:34

community wiki
3 revs
cat

If you're already detecting key strokes, why allow characters at all and throw errors around, when you can just silently ignore them, until you get the desired number? – Kebman Oct 31, 2020 at 14:27

1 @Kebman you could do that but it might be less obvious to the user what they can type – cat Nov 1, 2020 at 13:50

7

Use try-except to handle the error and repeat it again:

```
while True:
    try:
        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.")
    except Exception as e:
        print("please enter number")
```

Share

Improve this answer

Follow

edited Sep 11, 2020 at 14:11

community wiki
2 revs, 2 users 62%
behnaz.sheikhi

4 You are missing a `break` statement, and the `print("please enter number")` is unnecessary. – Georgy Sep 11, 2020 at 14:13

6

Building upon Daniel Q's and Patrick Artner's excellent suggestions, here is an even more generalized solution.

```
# Assuming Python3
import sys

class ValidationError(ValueError):  # thanks Patrick Artner
    pass

def validate_input(prompt, cast=str, cond=(lambda x: True), onerror=None):
    if onerror==None: onerror = {}
    while True:
        try:
            data = cast(input(prompt))
            if not cond(data): raise ValidationError
            return data
        except tuple(onerror.keys()) as e:  # thanks Daniel Q
            print(onerror[type(e)], file=sys.stderr)
```

I opted for explicit `if` and `raise` statements instead of an `assert`, because assertion checking may be turned off, whereas validation should always be on to provide robustness.

This may be used to get different kinds of input, with different validation conditions. For example:

```
# No validation, equivalent to simple input:
anystr = validate_input("Enter any string: ")

# Get a string containing only letters:
letters = validate_input("Enter letters: ",
    cond=str.isalpha,
    onerror={ValidationError: "Only letters, please!"})

# Get a float in [0, 100]:
percentage = validate_input("Percentage? ",
    cast=float, cond=lambda x: 0.0<=x<=100.0,
    onerror={ValidationError: "Must be between 0 and 100!",
             ValueError: "Not a number!"})
```

Or, to answer the original question:

```
age = validate_input("Please enter your age: ",
        cast=int, cond=lambda a:0<=a<150,
        onerror={ValidationError: "Enter a plausible age, please!",
                 ValueError: "Enter an integer, please!"})
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

Improve this answer

Follow

edited Dec 1, 2018 at 11:17

community wiki

3 revs

João Manuel Rodrigues

5

```
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

Improve this answer

Follow

answered Jun 23, 2016 at 10:34

community wiki

ojas mohril

4

Good question! You can try the following code for this. =)

This code uses ast.literal_eval() to **find the data type of the input** (`age`). Then it follows the following algorithm:

> 1. Ask user to input her/his `age`.
>
> 1.1. If `age` is `float` or `int` data type:
>
> - Check if `age>=18`. If `age>=18`, print appropriate output and exit.
> - Check if `0<age<18`. If `0<age<18`, print appropriate output and exit.
> - If `age<=0`, ask the user to input a valid number for age again, (*i.e.* go back to step 1.)
>
> 1.2. If `age` is not `float` or `int` data type, then ask user to input her/his age again (*i.e.* go back to step 1.)

Here is the code.

```
from ast import literal_eval

''' This function is used to identify the data type of input data.'''
def input_type(input_data):
    try:
        return type(literal_eval(input_data))
    except (ValueError, SyntaxError):
        return str

flag = True

while(flag):
    age = raw_input("Please enter your age: ")

    if input_type(age)==float or input_type(age)==int:
        if eval(age)>=18:
            print("You are able to vote in the United States!")
            flag = False
        elif eval(age)>0 and eval(age)<18:
            print("You are not able to vote in the United States.")
            flag = False
        else: print("Please enter a valid number as your age.")

    else: print("Sorry, I didn't understand that.")
```

Share

Improve this answer

Follow

edited Dec 18, 2018 at 6:54

community wiki
4 revs
Siddharth Satpathy

Try this one:-

3

```
def takeInput(required):
  print 'ooo or OOO to exit'
  ans = raw_input('Enter: ')

  if not ans:
      print "You entered nothing...!"
      return takeInput(required)

      ##  FOR Exit  ##
  elif ans in ['ooo', 'OOO']:
    print "Closing instance."
    exit()

  else:
    if ans.isdigit():
      current = 'int'
    elif set('[~!@#$%^&*()_+{}":/\']+$').intersection(ans):
      current = 'other'
    elif isinstance(ans,basestring):
      current = 'str'
    else:
      current = 'none'

  if required == current :
    return ans
  else:
    return takeInput(required)

## pass the value in which type you want [str/int/special character(as other )]
print "input: ", takeInput('str')
```

Share

Improve this answer

Follow

answered Apr 30, 2017 at 9:29

community wiki
Pratik Anand

2

Use "while" statement till user enter a true value and if the input value is not a number or it's a null value skip it and try to ask again and so on. In example I tried to answer truly your question. If we suppose that our age is between 1 and 150 then input value accepted, else it's a wrong value. For terminating program, the user can use 0 key and enter it as a value.

> Note: Read comments top of code.

```
# If your input value is only a number then use "Value.isdigit() == False".
# If you need an input that is a text, you should remove "Value.isdigit() ==
False".
def Input(Message):
    Value = None
    while Value == None or Value.isdigit() == False:
        try:
            Value = str(input(Message)).strip()
```

```
        except Exception:
            Value = None
    return Value

# Example:
age = 0
# If we suppose that our age is between 1 and 150 then input value accepted,
# else it's a wrong value.
while age <=0 or age >150:
    age = int(Input("Please enter your age: "))
    # For terminating program, the user can use 0 key and enter it as an a
value.
    if age == 0:
        print("Terminating ...")
        exit(0)

if age >= 18 and age <=150:
    print("You are able to vote in the United States!")
else:
    print("You are not able to vote in the United States.")
```

Share

Improve this answer

Follow

edited May 15, 2022 at 21:23

community wiki
6 revs, 2 users 99%
Saeed Zahedian Abroodi

1

To handle numeric input with validation and retries, consider using the `int_input` function from the `typed-input`[1] library (other functions like `float_input`, `decimal_input`, and `datetime_input` are also provided). It simplifies input validation, ensures type safety, and provides built-in support for error handling and custom messages—all without writing repetitive validation loops yourself.

Installation

Install the `typed-input` library via pip:

```
pip install typed-input
```

Here's a cleaner version of your program using `int_input`:

```
from typed_input import int_input

age = int_input(
    prompt="Please enter your age: ",
    min_value=1,  # Age must be a positive number
    type_error_message="Sorry, please enter a valid integer 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.")
```

Example Usage:

```
Please enter your age: sixty six
Sorry, please enter a valid integer age.
Please enter your age: -5
Error: Value must be at least 1.
Please enter your age: 26
You are able to vote in the United States!
```

Why use `typed-input`?

1. **Built-In Validation**: Automatically retries until the user provides valid input, ensuring user-friendly error handling.
2. **Customizable Prompts**: You can specify bounds (`min_value`, `max_value`) and custom error messages for better guidance.
3. **Cleaner Code**: Avoids you having to have repetitive input validation loops, saving development time.

---

[1] Disclosure: I am the author of `typed-input`.

Share
Improve this answer
Follow

answered Nov 21 at 12:28

community wiki
Sash Sinha

0

You can always apply simple if-else logic and add one more `if` logic to your code along with a `for` loop.

```
while True:
     age = int(input("Please enter your age: "))
     if (age >= 18)  :
         print("You are able to vote in the United States!")
     if (age < 18) & (age > 0):
         print("You are not able to vote in the United States.")
     else:
         print("Wrong characters, the input must be numeric")
         continue
```

This will be an infinite loo and you would be asked to enter the age, indefinitely.

Share
Improve this answer

edited Jul 1, 2019 at 14:17

community wiki
2 revs
Ep1c1aN

Follow

1 This doesn't really answer the question. The question was about getting a user input *until* they give a valid response, not *indefinitely*. – Georgy Jul 1, 2019 at 10:25

-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

Improve this answer

Follow

edited Jun 6, 2016 at 7:15

community wiki
2 revs
2Cubed

-1

You can write more general logic to allow user to enter only specific number of times, as the same use-case arises in many real-world applications.

```
def getValidInt(iMaxAttemps = None):
  iCount = 0
  while True:
    # exit when maximum attempt limit has expired
    if iCount != None and iCount > iMaxAttemps:
       return 0     # return as default value

    i = raw_input("Enter no")
    try:
       i = int(i)
    except ValueError as e:
       print "Enter valid int value"
    else:
       break

    return i

age = getValidInt()
# do whatever you want to do.
```

Share

answered Nov 3, 2016 at 7:49

community wiki

Improve this answer

Follow

Mangu Singh Rajpurohit

2 you forget to increase the iCount value after each loop – Hoai-Thu Vuong Mar 1, 2017 at 8:49

-1

You can make the input statement a while True loop so it repeatedly asks for the users input and then break that loop if the user enters the response you would like. And you can use try and except blocks to handle invalid responses.

```
while True:

    var = True

    try:
        age = int(input("Please enter your age: "))

    except ValueError:
        print("Invalid input.")
        var = False

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

The var variable is just so that if the user enters a string instead of a integer the program wont return "You are not able to vote in the United States."

Share

Improve this answer

Follow

answered Jan 3, 2018 at 0:59

community wiki
user9142415

-1

One more solution for using input validation using a customized `ValidationError` and a (optional) range validation for integer inputs:

```
class ValidationError(ValueError):
    """Special validation error - its message is supposed to be printed"""
    pass

def RangeValidator(text,num,r):
    """Generic validator - raises 'text' as ValidationError if 'num' not in
range 'r'."""
    if num in r:
        return num
    raise ValidationError(text)
```

```
def ValidCol(c):
    """Specialized column validator providing text and range."""
    return RangeValidator("Columns must be in the range of 0 to 3 (inclusive)",
                          c, range(4))

def ValidRow(r):
    """Specialized row validator providing text and range."""
    return RangeValidator("Rows must be in the range of 5 to 15(exclusive)",
                          r, range(5,15))
```

Usage:

```
def GetInt(text, validator=None):
    """Aks user for integer input until a valid integer is given. If provided,
    a 'validator' function takes the integer and either raises a
    ValidationError to be printed or returns the valid number.
    Non integers display a simple error message."""
    print()
    while True:
        n = input(text)
        try:
            n = int(n)

            return n if validator is None else validator(n)

        except ValueError as ve:
            # prints ValidationErrors directly - else generic message:
            if isinstance(ve, ValidationError):
                print(ve)
            else:
                print("Invalid input: ", n)


column = GetInt("Pleased enter column: ", ValidCol)
row = GetInt("Pleased enter row: ", ValidRow)
print( row, column)
```

Output:

```
Pleased enter column: 22
Columns must be in the range of 0 to 3 (inclusive)
Pleased enter column: -2
Columns must be in the range of 0 to 3 (inclusive)
Pleased enter column: 2
Pleased enter row: a
Invalid input:  a
Pleased enter row: 72
Rows must be in the range of 5 to 15(exclusive)
Pleased enter row: 9

9, 2
```


edited Nov 8, 2018 at 12:04

community wiki
3 revs
Patrick Artner

Follow

-1

Persistent user input using **recursive function**:

String

```
def askName():
    return input("Write your name: ").strip() or askName()

name = askName()
```

Integer

```
def askAge():
    try: return int(input("Enter your age: "))
    except ValueError: return askAge()

age = askAge()
```

and finally, the question requirement:

```
def askAge():
    try: return int(input("Enter your age: "))
    except ValueError: return askAge()

age = askAge()

responseAge = [
    "You are able to vote in the United States!",
    "You are not able to vote in the United States.",
][int(age < 18)]

print(responseAge)
```

Share

Improve this answer

Follow

edited Apr 15, 2019 at 13:56

community wiki
4 revs
Roko C. Buljan

-1

You can try to convert it to a integer, but ask the user to repeat if it doesn't work.

```
while True:
    age = input('Please enter your age: ')
    try:
        age_int = int(age)
        if age_int >= 18:
            print('You can vote in the United States!')
        else:
```

```
            print('You cannot vote in the United States.')
        break
    except:
        print('Please enter a meaningful answer.')
```

The while loop runs as long as the user has not inputted a meaningful answer, but breaks if it makes sense.

Share

Improve this answer

Follow

answered Feb 14, 2022 at 1:28

community wiki
neur0n-7

-2

Use `isdigit()` to check if a string represents a valid integer.

You could use a recursive function.

```
def ask():
    answer = input("Please enter amount to convert: ")
    if not answer.isdigit():
        print("Invalid")
        return ask()

    return int(answer)

Gdp = ask()
```

Or a while loop

```
while True:
    answer = input("Please enter amount to convert: ")
    if not answer.isdigit():
        print("Invalid")
        continue

    Gbp = int(answer)
```

Share

Improve this answer

Follow

edited Jun 30, 2021 at 0:03

community wiki
3 revs, 2 users 96%
Rein F

1 You're missing a `return` from the function. You `return` the recursive call, but that call returns `None`... And you `while` loop is infinite... – Tomerikoo May 4, 2021 at 19:16

@Tomerikoo It recursively asks until the answer is valid, which I think is what was asked. I meant to write it in a way where you can put any code *inside* the recursive function or while loop. This was actually written for a different question, which got marked as a duplicate to this one so I posted it here instead. – LevitatingBusinessMan May 5, 2021 at 16:15

1 What I mean is that you should test your code with some scenarios. In the first case, the `Gbp = int(answer)` should probably be `return int(answer)` and in the second there should probably be a `break` somewhere – Tomerikoo May 5, 2021 at 16:55

-7

Below code may help.

```
age=(lambda i,f: f(i,f))(input("Please enter your age: "),lambda i,f: i if
i.isdigit() else f(input("Please enter your age: "),f))
print("You are able to vote in the united states" if int(age)>=18 else "You are
not able to vote in the united states",end='')
```

If you want to have maximum tries, say 3, use below code

```
age=(lambda i,n,f: f(i,n,f))(input("Please enter your age: "),1,lambda i,n,f: i
if i.isdigit() else (None if n==3 else f(input("Please enter your age:
"),n+1,f)))
print("You are able to vote in the united states" if age and int(age)>=18 else
"You are not able to vote in the united states",end='')
```

Note: This uses recursion.

Share

Improve this answer

Follow

edited Aug 17, 2020 at 17:21

community wiki
3 revs
Liju

1 Don't use recursion to collect user input. Given enough retries, the app crashes. I don't understand the golfed code. Why not make it comprehensible? – ggorlen Jan 27, 2022 at 19:59

1 Teaching this to new users instead of a simple while-loop is obfuscatory and confusing. – smci Feb 10, 2022 at 21:07