Tkinter Validation

Summary: in this tutorial, you'll learn how to use the Tkinter validation to validate user inputs.

Introduction to the Tkinter validation

Tkinter validation relies on the three options that you can use for any input widget such as `Entry` (https://www.pythontutorial.net/tkinter/tkinter-entry/) widget:

- `validate` : specifies which type of event will trigger the validation.
- `validatecommand` : checks if a data is valid
- `invalidcommand` : executes when the data is invalid. In other words, it'll execute if the `validatecommand` returns `False` .

validate

The `validate` command can be one of the following string values:

'focus'	Validate whenever the widget gets or loses focus
'focusin'	Validate whenever the widget gets focus.

'focusout'	Validate whenever the widget loses focus
'key'	Validate whenever any keystroke changes the widget's contents.
'all'	Validate in all the above situations focusing, focusout, and key
'none'	Turn the validation off. This is the default. Please note that the string `'none'` is not the None (https://www.pythontutorial.net/advanced-python/python-none/) value in Python.

validatecommand

The `validatecommand` is a tuple that contains:

- A reference to a Tcl/tk function.
- Zero or more substitution codes specify the information that triggers the event you want to pass into the function.

To get a reference to a Tck/tk function, you pass a callable to the `widget.register()` method. It returns a string that you can use with the `validatecommand` .

The following table shows the substitution codes that you can use with the tuple:

`%d'`	Action code: 0 for an attempted deletion, 1 for an attempted insertion, or -1 if the callback was called for focus in, focus out, or a change to the `textvariable`.
`'%i'`	When the user attempts to insert or delete text, this argument will be the index of the beginning of the insertion or deletion. If the callback was due to focus in, focus out, or a change to the `textvariable`, the argument will be `-1`.
`'%P'`	The value that the text will have if the change is allowed.
`'%s'`	The text in the entry before the change.
`'%S'`	If the call was due to an insertion or deletion, this argument will be the text being inserted or deleted.
`'%v'`	The current value of the widget's `validate` option.

'%V'	The reason for this callback: one of 'focusin', 'focusout', 'key', or 'forced' if the textvariable was changed.
'%W'	The name of the widget.

The following example constructs a `validatecommand` that use the `self.validate()` method and `%P` substitution code:

```
vcmd = (self.register(self.validate), '%P')
```

invalidcommand

Like the `validatecommand`, the `invalidcommand` also requires the use of the `widget.register()` method and substitution code.

The following example returns a tuple that you can pass into the `invalidcommand` option:

```
ivcmd = (self.register(self.on_invalid),)
```

Tkinter validation example

We'll create a form that contains an email input. If you enter an invalid email address, it'll show an error message and change the text color of the email input to red. And we'll trigger the validation event when the focus is moving out of the entry.

Here's the complete program:

```
import tkinter as tk
from tkinter import ttk
import re


class App(tk.Tk):
    def __init__(self):
        super().__init__()
```

```
        self.title('Tkinter Validation Demo')

        self.create_widgets()

    def create_widgets(self):
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=3)
        self.columnconfigure(2, weight=1)

        # label
        ttk.Label(text='Email:').grid(row=0, column=0, padx=5, pady=5)

        # email entry
        vcmd = (self.register(self.validate), '%P')
        ivcmd = (self.register(self.on_invalid),)

        self.email_entry = ttk.Entry(self, width=50)
        self.email_entry.config(validate='focusout', validatecommand=vcmd, inv
        self.email_entry.grid(row=0, column=1, columnspan=2, padx=5)

        self.label_error = ttk.Label(self, foreground='red')
        self.label_error.grid(row=1, column=1, sticky=tk.W, padx=5)

        # button
        self.send_button = ttk.Button(text='Send').grid(row=0, column=4, padx=

    def show_message(self, error='', color='black'):
        self.label_error['text'] = error
        self.email_entry['foreground'] = color

    def validate(self, value):
        """
        Validat the email entry
        :param value:
        :return:
        """
```

```
            pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
            if re.fullmatch(pattern, value) is None:
                return False

            self.show_message()
            return True

        def on_invalid(self):
            """
            Show the error message if the data is not valid
            :return:
            """
            self.show_message('Please enter a valid email', 'red')


    if __name__ == '__main__':
        app = App()
        app.mainloop()
```

How it works:

First, create a validate command using the `self.validate()` method and `%P` substitution code:

```
vcmd = (self.register(self.validate), '%P')
```

Second, create the `invalidatecommand` that uses the `self.on_invalid` method:

```
ivcmd = (self.register(self.on_invalid),)
```

Third, configure the entry widget that uses `validation`, `validatecommand`, and `invalidatecommand`:

```
self.email_entry.config(validate='focusout', validatecommand=vcmd, invalidcomm
```

Fourth, define the `show_message()` method that changes the text of the `label_error` widget and the text color of the `email_entry` widget:

```
def show_message(self, error='', color='black'):
    self.label_error['text'] = error
    self.email_entry['foreground'] = color
```

Fifth, define the `validate()` method that validates the value of the `email_entry` .

```
def validate(self, value):
    """
    Validat the email entry
    :param value:
    :return:
    """
    pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    if re.fullmatch(pattern, value) is None:
        return False

    self.show_message()
    return True
```

The `validate()` method returns `True` if the input text is valid or `False` otherwise. In case the input text is a valid email address, call the `show_message()` to hide the error message and set the text color to black.

Tkinter will execute the `on_invalid()` method if the input text is not a valid email address.

Finally, define the `on_invalid()` method that shows an error message and set the text color of the `email_entry` widget to red.

```
def on_invalid(self):
    """
    Show the error message if the data is not valid
    :return:
```

```
        """
        self.show_message('Please enter a valid email', 'red')
```

Summary

- Tkinter uses the `validate`, `validatecommand`, and `invalidcommand` options on any input widget to validate data.
- Pass a callable to the `widget.register()` method to create a command for the `validatecommand` and `invalidcommand` options.
- `validationcommand` returns `True` if the data is valid or `False` otherwise.
- `invalidcommand` will execute if the data is not valid, or when the `validatecommand` return `False`.