ITEC 1150 Week 11 Chapter 8

INPUT VALIDATION USING PYINPUTPLUS

Input Validation – The Easy Way

- Last week, we learned about regular expressions and used them in the lab to perform input validation.
- Regular expressions are very useful and if you continue programming, you will need to know how to write them.
- However, for many common situations, there's a better (and easier!) way to do input validation:

Use a library

You'll often learn something the hard way, only to discover there's an easy way.

This is normal ☺

PyInputPlus
An Input and Validation Library

- PyInputPlus is an external library that is used to simplify the validation of user input.
- The input is a string you validate to see if contains the type of data required. You did a similar thing with regex and now you will see how to combine those with special input functions.
- Before proceeding, bookmark these resources:
- https://pyinputplus.readthedocs.io/ - the official documentation
- https://pypi.org/project/PyInputPlus/ - the project website
- https://github.com/asweigart/pyinputplus/blob/master/docs/index.rst - the project source

Adding A Library To Your Project

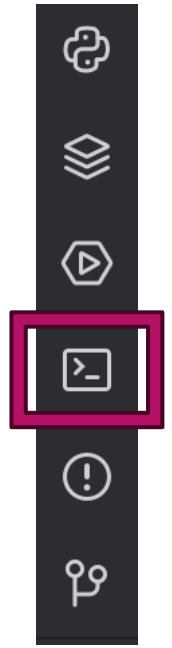
PYINPUTPLUS IS NOT A PART OF THE PYTHON STANDARD LIBRARY, SO WE HAVE TO ADD IT TO OUR PROJECT.

Two Options For Installing PyInputPlus

- The first option is to use the terminal and a Python tool called `pip` to install the library.
- This is usually very simple and quick.
- However:
  - if your system wasn't automatically configured properly when Python was installed, it won't work and it's time-intensive to troubleshoot.
  - if you try this approach and it doesn't work, use the second option.
- The second option is to use the PyCharm user interface to add a library to the project.
- This has a few more steps but relies less on Python being configured in a particular way.
- It's also a easier to troubleshoot.

Option 1 – Installing with `pip`

- In PyCharm, open the terminal window with the icon in the bottom-left toolbar, highlighted in the screen clip to the right.
- In the terminal, run the command `pip install pyinputplus`
- You should see some status messages scroll past, one of which will hopefully include "`Successfully installed pyinputplus-0.2.12`"
- If you see that, you can move on to the next section.
- If there are any error messages, continue in this section to install with the UI.

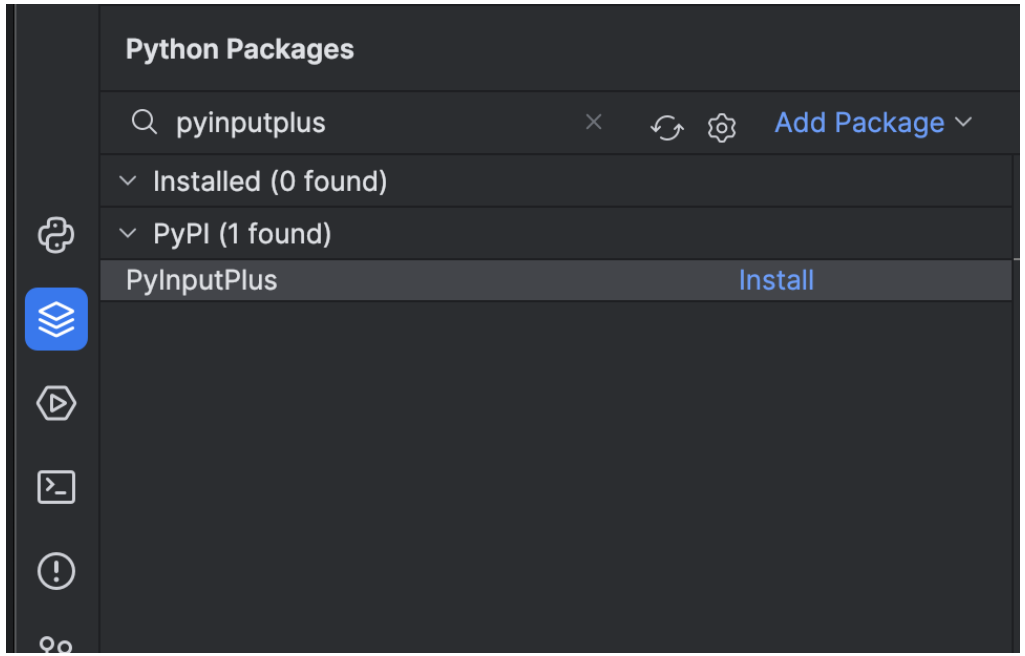


Option 2 – Installing Via The UI

- In your PyCharm project, open the Python Packages tool window by navigating in the menu to View → Tool Windows → Python Packages.
  - Alternatively, select the Python Packages icon in the bottom left area of the PyCharm window.
- The Python Packages tool window will open at the bottom of the PyCharm window.

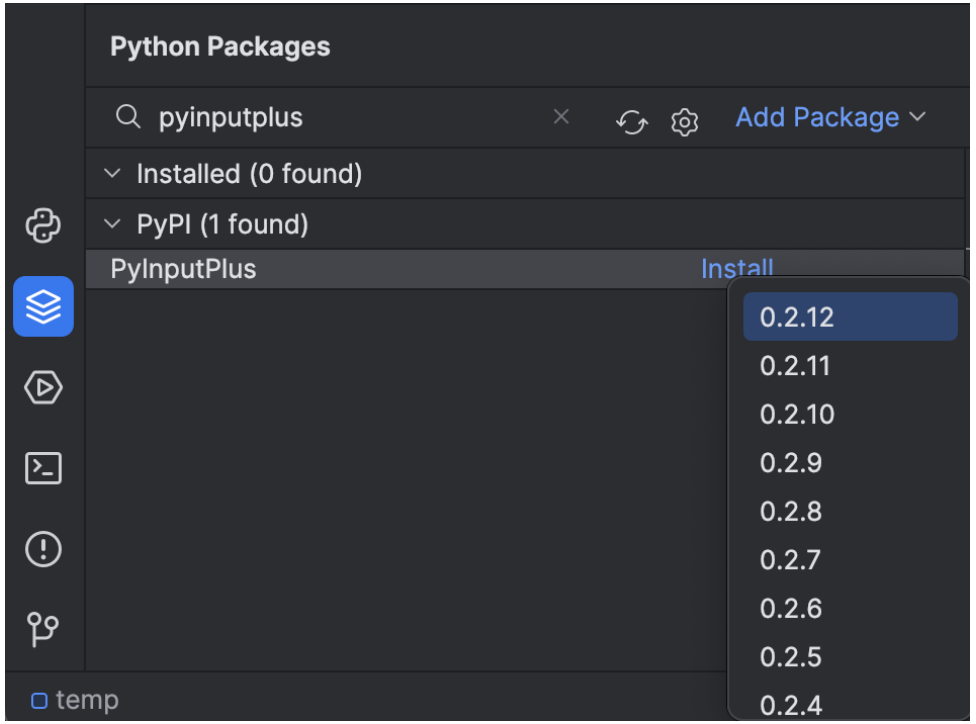
Option 2 (cont.): Search For A Package

- The top left corner of the Python Packages tool window contains a search box. Type 'pyinputplus' in that search box and you should see one result populate in the search results below the search box.



Option 2 (cont.): Install and Verify

- Click the 'Install' link next to the PyInputPlus package name and select the most recent version.
- PyCharm will take a moment to install the package and then the install link will change to the version number of the installed package.
- To verify installation, open the Python Console in PyCharm, and run the command `import pyinputplus`. If the command runs without error, PyInputPlus is installed correctly.



Using PyInputPlus

IMPORTING, FEATURES, AND USAGE

Importing PyInputPlus

- To use PyInputPlus in your program, simply import it:

```
1. import pyinputplus as pyip
```

- `as pyip` is an *alias*; it saves typing as we can now use `pyip.method()` instead of `pyinputplus.method()`.
- Create a file called ch8_exercises.py to run the lesson code.

A First Example Using PyInputPlus

```
1. import pyinputplus as pyip
2. myInt = pyip.inputInt("Enter a whole number between 1 and 10: ", min=1, max=10)
3. print('My int:', myInt)
```

- Run this code and experiment with different inputs.
- Notice how the parameters limit the input the program will accept?

Common PyInputPlus Functions

- `inputStr()` Is like the built-in `input()` function but has the general PyInputPlus features. You can also pass a custom validation function to it
- `inputNum()` Ensures the user enters a number and returns an int or float, depending on if the number has a decimal point in it
- `inputChoice()` Ensures the user enters one of the provided choices
- `inputMenu()` Is similar to `inputChoice()`, but provides a menu with numbered or lettered options
- `inputDatetime()` Ensures the user enters a date and time
- `inputYesNo()` Ensures the user enters a "yes" or "no" response
- `inputBool()` Is similar to `inputYesNo()`, but takes a "True" or "False" response and returns a Boolean value
- `inputEmail()` Ensures the user enters a valid email address
- `inputFilepath()` Ensures the user enters a valid file path and filename, and can optionally check that a file with that name exists
- `inputPassword()` Is like the built-in `input()`, but displays * characters as the user types so that passwords, or other sensitive information, aren't displayed on the screen
- FULL LIST ON THE LAST SLIDE!

Common PyInputPlus Function Parameters

- `prompt (string)`: The text to display before each prompt for user input. Required.
- `default (string)`: A default value to use should the user time out or exceed the number of tries to enter valid input. Should be combined with `limit (int)` or `timeout (int, float)`to have an effect.
- `blank (bool)`: If True, blank strings will be allowed as valid user input. False by default.
- `timeout (int, float)`: The number of seconds since the first prompt for input after which a TimeoutException is raised the next time the user enters input.
- `strip (bool, string)`: If True, whitespace is stripped from value. If a string, the characters in it are stripped from value. If None, nothing is stripped. Defaults to True.
- `limit (int)`: The number of tries the user has to enter valid input before the default value is returned.
- `allowRegexes (Sequence)`: A sequence of regex strings that will explicitly pass validation. Overrides blockRegexes. Defaults to None.
- `blockRegexes (Sequence)`: A sequence of regex strings or (regex_string, response_string) tuples that, if matched, will explicitly fail validation. Can be overridden by allowRegexes. Defaults to None.
- `applyFunction (Callable)`: An optional function that is passed the user's input and returns the new value to use as the input. This happens before validation.
- `postValidateApplyFunction (Callable)`: An optional function that is passed the user's input after it has passed validation and returns a transformed version for the input function to return.

`default` Function Parameter Example

- Experiment with the following function parameters by entering strings, blanks, and waiting for a few seconds before entering anything:

```
1. myInput = pyip.inputStr(prompt="Enter a string... ", default="A", limit=1, timeout=3)
2. print('My default if left blank:', myInput)
```

- The `default` parameter is used to set the default value if time runs out or the number of tries is exceeded.
- The `limit` parameter is used to specify the maximum number of tries the user has for entering a valid input.
- The `timeout` parameter is used to specify the maximum amount of time the user can take. If they take longer than that, the default value will be used no matter what they enter.
- A limit or timeout is needed with a default, or the default will never be applied.

`blockRegexes` and `allowRegexs` Parameters Example

- Experiment with the following function parameters by trying to find invalid inputs by entering different words starting with a capital M and with other capital letters:

```
1. myString = pyip.inputStr(prompt="Enter a string... ", blockRegexes=['^M.*'], allowRegexes=['^Minn.*'])
2. print('My string:', myString)
```

- The `blockRegexes` parameter is used to block any inputs that meet the provided regexes. The regexes should be provided as a list.
- The `allowRegexes` parameter is used to override the `blockRegexes` parameter; if the input matches any of these, it will be allowed even if it would've been blocked. The regexes should be provided as a list.

applyFunction and postValidateApplyFunction

- You can also pass functions to input methods. Whatever function you pass will be called with the string the user inputs:

```
1. def apply_format(string):
2.     return string.title()
3.
4. myInput = pyip.inputStr("Enter a name: ", postValidateApplyFunc=apply_format)
5. print('My converted string:', myInput)
```

- The `applyFunction` parameter will be called before validation happens, while the `postValidateApplyFunction` parameter will be called after validation happens.

Numeric Input Functions

- `inputNum()` Ensures the user enters a number and returns an int or float, depending on if the number has a decimal point in it
- `inputInt()` function is used to accept integer inputs
- `inputFloat()` function is used to accept float inputs

```
1. myInt = pyip.inputInt(prompt="Enter an integer... ")
2. print('My integer:' , myInt)
3. myNum = pyip.inputNum(prompt="Enter a number... ")
4. print('My number:' , myNum)
5. myFloat = pyip.inputFloat(prompt="Enter a float... ")
6. print('My float:' , myFloat)
```

- Experiment with the min and max parameters to set lower and upper bounds. The `min`, `max`, `greaterThan` and `lessThan` parameters are used to set the external bounds:

```
1. myInt = pyip.inputInt(prompt="Enter an Integer... ", min=3, lessThan=20) # a range from 3 to 19
2. print("My integer within range:", myInt)
```

The `inputChoice()` Function

- The `inputChoice()` function ensures the user enters one of the provided choices.
- Try this out with good and bad data:

```
1. myChoice = pyip.inputChoice(['Apple', 'Pomegranate', 'Tea-leaf'],
2.                             default = "BANANA!", limit=2)
3. print('My choice is:', myChoice)
```

The `inputMenu()` Function

- The `inputMenu()` function is similar to inputChoice() but provides a menu with numbered or lettered options.
- Try this out with good and bad data:

```
1. myMenuChoice = pyip.inputMenu(['Apple', 'Pomegranate', 'Tea-leaf'],
2.                               default = 'Apple', limit=2)
3. print('My choice is:', myMenuChoice)
```

- Notice, if you provide a default, it must be a valid option to avoid an exception error.
- Try the *numbered = True* or *lettered = True* parameters in preference to a simple * in the menu.

```
1. myChoice = pyip.inputMenu(['Apple', 'Pomegranate', 'Tea-leaf'],
2.                           default = "Apple", limit=2, lettered=True)
3. print('My Choice is:', myChoice)
```

Date and Time Input Functions

- The function `inputDateTime()` ensures the user enters a valid date and time:

```
1. myDateTime = pyip.inputDatetime("Enter a date and time ",
2.                                 formats=('%m/%d/%y %H:%M:%S', '%m.%d.%Y %H:%M:%S'))
3. print('My Date and Time are:', myDateTime)
```

- If you just need a date, use the `inputDate()` function:

```
1. myDate = pyip.inputDate("Enter a valid date")
2. print('My Date is:', myDate)
```

- If you just need a time, use the `inputTime()` function:

```
1.  myTime = pyip.inputTime("Enter a valid time")
2.  print('My Date is:', myTime)
```

- See https://www.w3schools.com/python/gloss_python_date_format_codes.asp for info on date format codes.

Yes/No and True/False Input Functions

- The `inputYesNo()` function ensures the user enters a "yes" or "no" response. It is not case sensitive so respond with YES or YeS or nO or No etc. It's great for asking to end a program or not.

```
1.  myYesNo = pyip.inputYesNo("Would you like to enter more data?")
2.  print('My response is:', myYesNo)
```

- The `inputBool()` function is similar to `inputYesNo()`, but takes a "True" or "False" response and returns a Boolean value:

```
1. myBool = pyip.inputBool("Would you like to enter more data?")
2. print('My response is:', myBool)
```

The `inputEmail()` Function

- The **`inputEmail()`** function ensures the user enters a validly structured email address.
- Obviously, it can't verify that the email address is actually real. Without sending an email to that address you will never know!

```
1. myEmail = pyip.inputEmail('Please enter a valid email address')
2. print("My email address is:", myEmail)
```

… and there is so much more!

- If you work through the PPT and book Chapter 8 exercises, while thinking of all the programs you have already written, you will probably realize that using special validation functions can reduce the amount of code you would otherwise need to validate input.
- You will find programs quicker and easier to code!
- This introduction gets you started and will help you see the difference between writing a lot of code to validate input, versus using a pre-written function or creating your own function that includes a regular expression to do the same thing.

```
inputBool
inputChoice
inputCustom
inputDate
inputDatetime
inputDayOfMonth
inputDayOfWeek
inputEmail
inputFilename
inputFilepath
inputFloat
inputIP
inputInt
inputMenu
inputMonth
inputNum
inputPassword
inputRegex
inputRegexStr
inputStr
inputTime
inputURL
inputUSState
inputYesNo
inputZip
```