Table of contents

---

1. import PyInputPlus

(go to top)

https://pyinputplus.readthedocs.io/ (https://pyinputplus.readthedocs.io/).

In [1]:
```
import pyinputplus
```

- PyInputPlus contains functions similar to input() for several kinds of data: numbers, dates, email addresses, and more.
- If the user ever enters invalid input, such as a badly formatted date or a number that is outside of an intended range,
- PyInputPlus will reprompt them for input just like with try and except.
- PyInputPlus also has other useful features like a limit for the number of times it reprompts users and a timeout if users are required to respond within a time limit.

- `inputStr()` Is like the built-in input() function but has the gen- eral 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 num- bered 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

In [1]:
```
import pyinputplus as pyip
```

In [7]:
```
response = pyip.inputNum()
print('you entered:', response)
```

```
five
'five' is not a number.
5
you entered: 5
```

2. Add a prompt

(go to top)

In [11]:
```
response = pyip.inputInt(prompt = 'Enter a number: ')
print('you entered:', response)
```

```
Enter a number: dog
'dog' is not an integer.
Enter a number: 5.5
'5.5' is not an integer.
Enter a number: 5
you entered: 5
```

In [ ]:
```

```

3. The min, max, greaterThan, and lessThan Keyword Arguments

(go to top)

In [14]:
```
# min
response = pyip.inputNum('Enter num: ', min=4)
print('you entered:', response)
```

```
Enter num: 3
Number must be at minimum 4.
Enter num: 5
you entered: 5
```

In [17]:
```
# max
response = pyip.inputNum('Enter num: ', max=4)
print('you entered:', response)
```

```
Enter num: 5
Number must be at maximum 4.
Enter num: 3
you entered: 3
```

In [15]:
```
# greaterThan
response = pyip.inputNum('Enter num: ', greaterThan=4)
print('you entered:', response)
```

```
Enter num: 4
Number must be greater than 4.
Enter num: 6
you entered: 6
```

In [16]:
```
# lessThan
response = pyip.inputNum('>', min=4, lessThan=6)
print('you entered:', response)
```

```
>3
Number must be at minimum 4.
>7
Number must be less than 6.
>5
you entered: 5
```

In [ ]:
```

```

---

4. The blank Keyword Argument

- By default, blank input isn't allowed unless the blank keyword argument is set to True:

(go to top)

In [19]:
```
response = pyip.inputNum('Enter num: ')
print('you entered:', response)
```

```
Enter num:
Blank values are not allowed.
Enter num: 1
you entered: 1
```

In [20]:
```
response = pyip.inputNum('Enter num: ', blank = True)
print('you entered:', response)
```

```
Enter num:
you entered:
```

5. The limit, timeout, and default Keyword Arguments

- By default, the PyInputPlus functions will continue to ask the user for valid input forever (or for as long as the program runs).
- Use the `limit` keyword argument to determine how many attempts a PyInputPlus function will make to receive valid input before giving up,
- Use the `timeout` keyword argument to determine how many seconds the user has to enter valid input before the PyInputPlus func- tion gives up.
- Pass a `default` keyword argument, and the function returns the default value instead of raising an exception.

(go to top)

In [7]:
```
# enter the wrong input twice

response = pyip.inputNum(limit = 2, timeout = 10, default = 'N/A')

print('response is: ', response)
```

```
hey
'hey' is not a number.
you
'you' is not a number.
response is:  N/A
```

In [11]:
```
# enter a number after 10 seconds of waiting

response = pyip.inputNum(limit = 2, timeout = 10, default = 'N/A')

print('response is: ', response)
```

```
5
response is:  N/A
```

In [10]:
```
# enter a number after 10 seconds of waiting

# response = pyip.inputNum(timeout=10)
```

In [9]:
```
# give a wrong input twice

#response = pyip.inputNum(limit=2)
```

In [ ]:
```

```

6. `allowRegexes` and `blockRegexes` Keywords

- The `allowRegexes` and `blockRegexes` keyword arguments take a list of regular expression strings to determine what the PyInputPlus function will accept or reject as valid input.
- For example, `inputNum()` below will accept Roman numerals in addition to the usual numbers:
  - Of course, this regex affects only what letters the `inputNum()` function will accept from the user; the function will still accept Roman numerals with invalid ordering such as `'XVX' or 'MILLI'` because the `r'(I|V|X|L|C|D|M)+'` regular expression accepts those strings.

(go to top)

In [13]:
```
response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+', r'zero'])
print('response is: ', response)
```

```
XLII
response is:  XLII
```

In [19]:
```
response = pyip.inputNum(allowRegexes=[r'(i|v|x|l|c|d|m)+', r'zero'])
print('response is: ', response)
```

```
xlii
response is:  xlii
```

In [16]:
```
response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+', r'zero'])
print('response is: ', response)
```

```
zero
response is:  zero
```

In [17]:
```
response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+'])
print('response is: ', response)
```

```
zero
'zero' is not a number.
5
response is:  5
```

In [24]:
```
# won't accept even numbers
response = pyip.inputNum(blockRegexes=[r'[02468]$'])
print('response is: ', response)
```

```
2
This response is invalid.
4
This response is invalid.
6
This response is invalid.
24
This response is invalid.
3
response is:  3
```

- If you specify both an allowRegexes and blockRegexes argument, the allow list overrides the block list.

In [26]:
```
response = pyip.inputStr(allowRegexes=[r'caterpillar', 'category'], bl
print('response is: ', response)
```

```
cat
This response is invalid.
catapult
This response is invalid.
catastrophe
This response is invalid.
caterpillar
response is:  caterpillar
```

7. Passing a Custom Validation Function to `inputCustom()`

(go to top)

You can write a function to perform `your own custom validation logic by passing the function to inputCustom()` . For example, say you want the user to enter `a series of digits that adds up to 10.`

There is no pyinputplus inputAddsUpToTen() function, but you can create your own function that:

- Accepts a single string argument of what the user entered
- Raises an exception if the string fails validation
- Returns None (or has no return statement) if inputCustom() should return the string unchanged
- Returns a non-None value if inputCustom() should return a different string from the one the user entered
- Is passed as the first argument to inputCustom()

For example, we can create our own addsUpToTen() function, and then pass it to inputCustom().

- Note that the function call looks `like inputCustom(addsUpToTen) and not inputCustom(addsUpToTen())`
- because `we are passing the addsUpToTen() function itself to inputCustom(), not calling addsUpToTen() and passing its return value.`

In [35]:
```
#accepts a string of numbers
def addsUpToTen(strNums):
    numbers = list(strNums)

    for i, num in enumerate(numbers):
        numbers[i] = int(num)

    if sum(numbers) != 10:
        raise Exception('The numbers should add up to 10 not %s' %(sum

    return int(strNums)
```

In [37]:
```
userInput = input()
addsUpToTen(userInput)
```

```
245
```

```
---------------------------------------------------------------------------
------
Exception                                 Traceback (most recent call
last)
<ipython-input-37-fe0723984fd3> in <module>
      1 userInput = input()
----> 2 addsUpToTen(userInput)

<ipython-input-35-1929651c8ad4> in addsUpToTen(strNums)
      7
      8     if sum(numbers) != 10:
----> 9         raise Exception('The numbers should add up to 10 not
%s' %(sum(numbers)))
     10
     11     return int(strNums)

Exception: The numbers should add up to 10 not 11
```

In [38]:
```
response = pyip.inputCustom(addsUpToTen)
```

```
245
The numbers should add up to 10 not 11
649
The numbers should add up to 10 not 19
55
```

In [39]:
```
response
```

Out[39]:
```
55
```

8.inputYesNo()

(go to top)

In [43]:
```
pyip.inputYesNo?
```

In [44]:
```
response = pyip.inputYesNo('Do you want money: ')
print(response)
```

```
Do you want money: y
yes
```

In [45]:
```
response = pyip.inputYesNo('Do you want money: ')
if response == 'yes': # y or YES or yes
    print('here you go: $$$')
else:
    print('okay bye')
```

```
Do you want money: y
here you go: $$$
```

In [55]:
```
response = pyip.inputYesNo('Do you want money: ', yesVal = 'yeahyeah',
if response == 'yeahyeah': # y or yeah yeah
    print('here you go: $$$')
elif response == 'noway':
    print('okay bye')
```

```
Do you want money: yes
'yes' is not a valid yeahyeah/noway response.
Do you want money: YES
'YES' is not a valid yeahyeah/noway response.
Do you want money: y
here you go: $$$
```

In [56]:
```
response = pyip.inputYesNo('Do you want money: ', yesVal = 'yeahyeah',
if response == 'yeahyeah': # y or yeah yeah
    print('here you go: $$$')
elif response == 'noway':
    print('okay bye')
```

```
Do you want money: nope
'nope' is not a valid yeahyeah/noway response.
Do you want money: no
'no' is not a valid yeahyeah/noway response.
Do you want money: n
okay bye
```

In [58]:
```
# try it out

response = pyip.inputYesNo('Do you want money: ', yesVal = 'yeahyeah',
if response == 'yeahyeah': # y or yeah yeah
    print('here you go: $$$')
elif response == 'noway':
    print('okay bye')
```

```
Do you want money: yeahyeah
here you go: $$$
```

In [ ]:

9. Multiplication Quiz

(go to top)

In [59]:
```
import pyinputplus as pyip, random, time
```

In [71]:
```
noOfQuestions = 10
correctAnswers = 0

for question in range(noOfQuestions):
    num1 = random.randint(5,9)
    num2 = random.randint(4,9)

    prompt = 'Q%s : %s x %s = ' % (question, num1, num2)

    # Right answers are handled by allowRegexes.
     # Wrong answers are handled by blockRegexes, with a custom messag
    try:
        pyip.inputStr(prompt,
                      allowRegexes = ['^%s$' % (num1 * num2)],
                      blockRegexes = [('.*', 'Incorrect!')],
                      timeout = 8,
                      limit = 3)
    except pyip.TimeoutException:
        print('Out of time!')
    except pyip.RetryLimitException:
        print('Out of tries!')
    else:
        # This block runs if no exceptions were raised in the try bloc
        print('Correct!')
        correctAnswers += 1
        time.sleep(1) # Brief pause to let user see the result.

print('Score: %s / %s' % (correctAnswers, noOfQuestions))
```

```
Q0 : 9 x 4 = 36
Correct!
Q1 : 6 x 5 = 30
Correct!
Q2 : 6 x 6 = 36
Correct!
Q3 : 5 x 8 = 40
Correct!
Q4 : 7 x 6 = 42
Correct!
Q5 : 9 x 6 = 54
```

```
Correct!
Q6 : 9 x 7 = 63
Correct!
Q7 : 8 x 7 = 58
Incorrect!
Q7 : 8 x 7 = 56
Out of time!
Q8 : 5 x 6 = 30
Correct!
Q9 : 5 x 8 = 40
Correct!
Score: 9 / 10
```

In [ ]:

In [ ]:

In [ ]: s

In [ ]:

In [ ]:

10. Title

(go to top)

11. Title

(go to top)

12. Title

(go to top)

13. Title

(go to top)

14. Title

(go to top)

15. Title

(go to top)

16. Title

(go to top)

17. Title

(go to top)

18. Title

(go to top)

19. Title

(go to top)

20. Title

(go to top)

In [66]:
```
pyip.inputStr?
```

In [ ]:
```

```