from pathlib import Path # Here “P” should be capitalized

Path.cwd()

os.chdir()

Path.home()

os.makedirs()

**Dictionary Functions**

someDic.item() # will get all keys & values in dic

someDic.values() or keys() # will get values or keys to corresponding keys or values

**The get() method**

**someDic.get(‘somekey’, “fall back value”)**

**# this methods first checks if the key exits in Dic**

1. **If yes then it returns its value**
2. **If not then it return fall back value.**

**Pretty printing**

pprint.pprint(somedic) # will print Dic tems on every new line

print(pprint.pformat(somedic) # Will convert Dic into A single string

**The Setdefault() method**

**someDic.get(‘somekey’, “default value”)**

**# this methods first checks if the key exits in Dic**

1. **If yes then it returns its value**
2. **If not then it adds the key & the default value to it. Then returns Default value.**

**6. Manipulating String**

**The upper(), lower(), isupper() & islower() method :-**

veriable\_or\_string.upper or lower()

veriable\_or\_string.isupper or islower()

**The isx() method:-**

isalpha() Returns True if the string consists only of letters and isn’t blank

isalnum() Returns True if the string consists only of letters and numbers and is not blank

isdecimal() Returns True if the string consists only of numeric characters and is not blank

isspace() Returns True if the string consists only of spaces, tabs, and newlines and is not blank

istitle() Returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters

Use like – “veriable\_or\_string.isx()

**The strartswith() & endswith() method:-**

veriable\_or\_string.startswith()\_or\_endswith()

**The join() & split() method:-**

String.join([some\_list])

String.split(‘string to separate’) # The “string” used to separate will be erased.

Splitting string with partition method:-

>>> 'Hello, world!'.partition('w')

***Output-***('Hello, ', 'w', 'orld!')

>>> 'Hello, world!'.partition('XYZ')

***output-*** ('Hello, world!', '', '') # xyz isn’t in given string

Justifying Text with the rjust(), ljust(), and center() Methods

>>> 'Hello'.rjust(10)

***output-*** ' Hello'

>>> 'Hello'.ljust(10)

***output-*** 'Hello '

>>> 'Hello'.rjust(20, '\*')

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hello'

Removing Whitespace with the strip(), rstrip(), and lstrip() Methods

string or variable containing string.strip() # Will strip white space from both side of string (start & end) because there is no value in bracket.

Another example

>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'

>>> spam.strip('ampS')

'BaconSpamEggs'

Numeric Values of Characters with the ord() and chr() Functions

>>> ord('A')

65

>>> chr(65)

'A'

Copying and Pasting Strings with the pyperclip Module

>>> import pyperclip

>>> pyperclip.copy('Hello, world!')

>>> pyperclip.paste()

'Hello, world!'

**7 PATTERN MATCHING WITH REGULAR EXPRESSIONS**

1. To use regular expressions ( Regex) You have to first “import re”

2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)

3. Pass the string you want to search into the Regex object’s search() method. This returns a Match object.

4. Call the Match object’s group() method to return a string of the actual matched text.

**Grouping with Parentheses**

>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

>>> mo = phoneNumRegex.search('My number is 415-555-4242.')

>>> mo.group(1)

'415'

>>> mo.group(2)

'555-4242'

>>> mo.group(0)

'415-555-4242'

>>> mo.group()

'415-555-4242'

# we can also use multiple-assinment work with groups() method

. ^ $ \* + ? { } [ ] \ | ( )

If you want to detect these characters as part of your text pattern, you need to escape them with a backslash:

\. \^ \$ \\* \+ \? \{ \} \[ \] \\ \| \( \)

**Matching Multiple Groups with the Pipe(|)**

>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')

>>> mo = batRegex.search('Batmobile lost a wheel')

>>> mo.group()

'Batmobile'

>>> mo.group(1)

'mobile'

**Optional Matching with the Question Mark**

>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')

>>> mo1 = phoneRegex.search('My number is 415-555-4242')

>>> mo1.group()

'415-555-4242'

>>> mo2 = phoneRegex.search('My number is 555-4242')

>>> mo2.group()

'555-4242'

**Matching Zero or More with the Star \***

>>> batRegex = re.compile(r'Bat(wo)\*man')

>>> mo1 = batRegex.search('The Adventures of Batman')

>>> mo1.group()

'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')

>>> mo2.group()

'Batwoman'

>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')

>>> mo3.group()

'Batwowowowoman'

**Matching One or More with the Plus**

>>> batRegex = re.compile(r'Bat(wo)+man')

>>> mo1 = batRegex.search('The Adventures of Batwoman')

>>> mo1.group()

'Batwoman'

>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')

>>> mo2.group()

'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')

>>> mo3 == None

True

**Matching Specific Repetitions with Braces**

>>> haRegex = re.compile(r'(Ha){3}')

>>> mo1 = haRegex.search('HaHaHa')

>>> mo1.group()

'HaHaHa'

>>> mo2 = haRegex.search('Ha')

>>> mo2 == None

True

**Greedy and Non-greedy Matching**

>>> greedyHaRegex = re.compile(r'(Ha){3,5}')

>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')

>>> mo1.group()

'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')

>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')

>>> mo2.group()

'HaHaHa'

**The findall() Method**

>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups

>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')

['415-555-9999', '212-555-0000']

**Character Classes**

|  |  |
| --- | --- |
| Short hand Character class | Reprasents |
| \d | Any numeric digit from 0 to 9. |
| \D | Any character that is not a numeric digit from 0 to 9. |
| \w | Any letter, numeric digit, or the underscore character. (Think of this as matching “word” characters.) |
| \W | Any character that is not a letter, numeric digit, or the underscore character. |
| \s | Any space, tab, or newline character. (Think of this as matching “space” characters.) |
| \S | Any character that is not a space, tab, or newline. |

**Making Your Own Character Classes**

>>> vowelRegex = re.compile(r'[aeiouAEIOU]')

>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')

['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']

# You can also include ranges of letters or numbers by using a hyphen. For example, the character class [azA-Z0-9] will match all lowercase letters, uppercase letters, and numbers.

the character class [0-5.] will match digits 0 to 5 and a period. You do not need to write it as [0-5\.].

Negative Caret (^) character class

>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')

>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')

['R', 'b', 'C', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd', '.', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']

**The Caret and Dollar Sign Characters**

>>> wholeStringIsNum = re.compile(r'^\d+$')

>>> wholeStringIsNum.search('1234567890')

<re.Match object; span=(0, 10), match='1234567890'>

>>> wholeStringIsNum.search('12345xyz67890') == None

True

>>> wholeStringIsNum.search('12 34567890') == None

True

**The Wildcard Character**

The **. (or dot)** character in a regular expression is called a wildcard and will match any character except for a newline.

>>> atRegex = re.compile(r'.at')

>>> atRegex.findall('The cat in the hat sat on the flat mat.')

['cat', 'hat', 'sat', 'lat', 'mat']

**Matching Everything with Dot-Star (.\*)**

**Matching Newlines with the Dot Character (re.DOTALL)**

>>> noNewlineRegex = re.compile('.\*')

>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent. \nUphold the law.').group()

'Serve the public trust.'

>>> newlineRegex = re.compile('.\*', re.DOTALL)

>>> newlineRegex.search('Serve the public trust.\nProtect the innocent. \nUphold the law.').group()

'Serve the public trust.\nProtect the innocent.\nUphold the law.'

**Review of Regex Symbols**

This chapter covered a lot of notation, so here’s a quick review of what you learned about basic regular expression syntax:

* The ? matches zero or one of the preceding group.
* The \* matches zero or more of the preceding group.
* The + matches one or more of the preceding group.
* The {n} matches exactly n of the preceding group.
* The {n,} matches n or more of the preceding group.
* The {,m} matches 0 to m of the preceding group.
* The {n,m} matches at least n and at most m of the preceding group.
* {n,m}? or \*? or +? performs a non-greedy match of the preceding group.
* ^spam means the string must begin with spam.
* spam$ means the string must end with spam.
* The . matches any character, except newline characters.
* \d, \w, and \s match a digit, word, or space character, respectively.
* \D, \W, and \S match anything except a digit, word, or space character, respectively.
* [abc] matches any character between the brackets (such as a, b, or c).
* [^abc] matches any character that isn’t between the brackets.

**Case-Insensitive Matching**

Normally, regular expressions match text with the exact casing you specify.

To make your regex case-insensitive, you can pass re.IGNORECASE or re.I as a second argument to re.compile().

>>> robocop = re.compile(r'robocop', re.I)

>>> robocop.search('RoboCop is part man, part machine, all cop.').group()

'RoboCop'

>>> robocop.search('ROBOCOP protects the innocent.').group()

'ROBOCOP'

>>> robocop.search('Al, why does your programming book talk about robocop so much?').group()

'robocop'

**Substituting Strings with the sub() Method**

>>> namesRegex = re.compile(r'Agent \w+')

>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.') 'CENSORED gave the secret documents to CENSORED.'

Sometimes you may need to use the matched text itself as part of the substitution. In the first argument to sub(), you can type \1, \2, \3, and so on, to mean “Enter the text of group 1, 2, 3, and so on, in the substitution.”

>>> agentNamesRegex = re.compile(r'Agent (\w)\w\*')

>>> agentNamesRegex.sub(r'\1\*\*\*\*', 'Agent Alice told Agent Carol that Agent Eve knew Agent Bob was a double agent.')

‘A\*\*\*\* told C\*\*\*\* that E\*\*\*\* knew B\*\*\*\* was a double agent.'

**Managing Complex Regexes with re.whitespace**

phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)\d{4} (\s\*(ext|x|ext.)\s\*\d{2,5})?)')

Instead of this

phoneRegex = re.compile(r'''(

(\d{3}|\(\d{3}\))? # area code

(\s|-|\.)? # separator

\d{3} # first 3 digits

(\s|-|\.) # separator

\d{4} # last 4 digits

(\s\*(ext|x|ext.)\s\*\d{2,5})? # extension

)''', re.VERBOSE)

Use this

**Combining re.IGNORECASE, re.DOTALL, and re.VERBOSE with Pipe(|) Character**

>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL | re.VERBOSE)

**Input Validation with pyinputplus Module**

>>> import pyinputplus

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

**>>> response = input('Enter a number: ')**

**Enter a number: 42**

**>>> response**

**'42'**

**>>> import pyinputplus as pyip**

**>>> response = pyip.inputInt(prompt='Enter a number: ')**

**Enter a number: cat**

**'cat' is not an integer.**

**Enter a number: 42**

**>>> response**

**42**

**The min, max, greaterThan, and lessThan Keyword Arguments for inputNum(), inputInt(), and inputFloat() functions**

>>> import pyinputplus as pyip

>>> response = pyip.inputNum('Enter num: ', min=4)

Enter num:3

Input must be at minimum 4.

Enter num:4

>>> response

4

>>> response = pyip.inputNum('Enter num: ', greaterThan=4)

Enter num: 4

Input must be greater than 4.

Enter num: 5

>>> response

5

>>> response = pyip.inputNum('>', min=4, lessThan=6)

Enter num: 6

Input must be less than 6.

Enter num: 3

Input must be at minimum 4.

Enter num: 4

>>> response

4

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

**The limit, timeout, and default Keyword Arguments**

>>> import pyinputplus as pyip

>>> response = pyip.inputNum(limit=2)

blah

'blah' is not a number.

Enter num: number

'number' is not a number.

Traceback (most recent call last):

--snip--

pyinputplus.RetryLimitException

>>> response = pyip.inputNum(timeout=10)

42 (entered after 10 seconds of waiting)

Traceback (most recent call last):

--snip--

pyinputplus.TimeoutException

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

hello

'hello' is not a number.

world

'world' is not a number.

>>> response

'N/A'

**The allowRegexes and blockRegexes Keyword Arguments**

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.

>>> import pyinputplus as pyip

>>> response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+', r'zero'])

XLII

>>> response

'XLII'

>>> response = pyip.inputNum(allowRegexes=[r'(i|v|x|l|c|d|m)+', r'zero'])

xlii

>>> response

'xlii'

>>> import pyinputplus as pyip

>>> response = pyip.inputNum(blockRegexes=[r'[02468]$'])

42

This response is invalid.

44

This response is invalid.

43

>>> response

43

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

>>> import pyinputplus as pyip

>>> response = pyip.inputStr(allowRegexes=[r'caterpillar', 'category'],

blockRegexes=[r'cat'])

cat

This response is invalid.

catastrophe

This response is invalid.

category

>>> response

'category'

**Passing a Custom Validation Function to inputCustom()**

**>>> import pyinputplus as pyip**

**>>> def addsUpToTen(numbers):**

**... numbersList = list(numbers)**

**... for i, digit in enumerate(numbersList):**

**... numbersList[i] = int(digit)**

**... if sum(numbersList) != 10:**

**... raise Exception('The digits must add up to 10, not %s.' %**

**(sum(numbersList)))**

**... return int(numbers) # Return an int form of numbers.**

**...**

**>>> response = pyip.inputCustom(addsUpToTen) # No parentheses after**

**addsUpToTen here.**

**123**

**The digits must add up to 10, not 6.**

**1235**

**The digits must add up to 10, not 11.**

**1234**

**>>> response # inputStr() returned an int, not a string.**

**1234**

**>>> response = pyip.inputCustom(addsUpToTen)**

**hello**

**invalid literal for int() with base 10: 'h'**

**55**

**>>> response**

The inputCustom() function also supports the general PyInputPlus features, such as the blank, limit, timeout, default, allowRegexes, and blockRegexes keyword arguments.

**Reading & Writing Files**

**Backslash on Windows and Forward Slash on macOS and Linux**

the following code joins names from a list of filenames to the end of a folder’s name:

>>> from pathlib import Path

>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']

>>> for filename in myFiles:

print(Path(r'C:\Users\Al', filename))

C:\Users\Al\accounts.txt

C:\Users\Al\details.csv

C:\Users\Al\invite.docx

**Using the / Operator to Join Paths**

>>> from pathlib import Path

>>> Path('spam') / 'bacon' / 'eggs'

WindowsPath('spam/bacon/eggs')

>>> Path('spam') / Path('bacon/eggs')

WindowsPath('spam/bacon/eggs')

>>> Path('spam') / Path('bacon', 'eggs')

WindowsPath('spam/bacon/eggs')

>>> homeFolder = r'C:\Users\Al'

>>> subFolder = 'spam'

>>> homeFolder + '\\' + subFolder

'C:\\Users\\Al\\spam'

>>> homeFolder = Path('C:/Users/Al')

>>> subFolder = Path('spam')

>>> homeFolder / subFolder

WindowsPath('C:/Users/Al/spam')

>>> str(homeFolder / subFolder)

'C:\\Users\\Al\\spam'