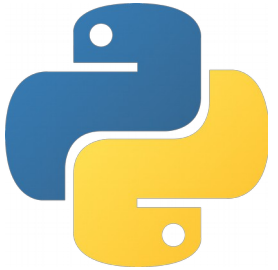
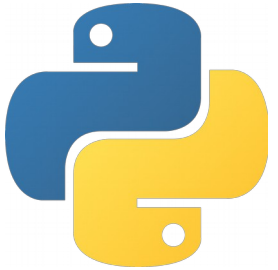
Python Insiderwissen



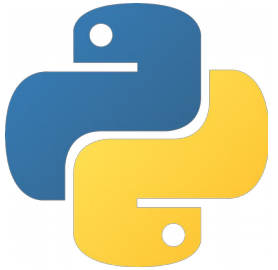
- URLs im Standardbrowser öffnen – webbrowser
  - Das Modul webbrowser dient dazu, eine Website im Standardbrowser des gerade verwendeten Systems zu öffnen
  - Die Wesentlichste Methode ist open
  - open(url[, new[, autoraise]])
    - Diese Funktion öffnet die URL url im Standardbrowser des Systems
    - Für den Parameter new kann eine ganze Zahl zwischen 0 und 2 übergeben werden

Python Insiderwissen



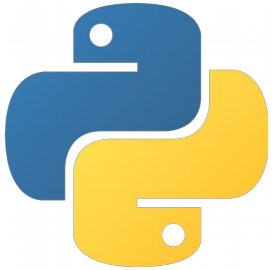
- URLs im Standardbrowser öffnen – webbrowser
  - Das Modul webbrowser dient dazu, eine Website im Standardbrowser des gerade verwendeten Systems zu öffnen
  - Die Wesentlichste Methode ist open
  - open(url[, new[, autoraise]])
    - Dabei bedeutet ein Wert von
      - 0, dass die URL nach Möglichkeit in einem bestehenden Browserfenster geöffnet wird (default)
      - 1, dass die URL in einem neuen Browserfenster geöffnet werden soll
      - 2, dass die URL nach Möglichkeit in einem neuen Tab eines bestehenden Browserfensters geöffnet werden soll

Python Insiderwissen



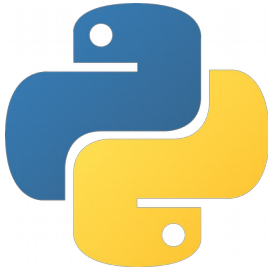
- URLs im Standardbrowser öffnen – webbrowser
  - Das Modul webbrowser dient dazu, eine Website im Standardbrowser des gerade verwendeten Systems zu öffnen
  - Die Wesentlichste Methode ist open
  - open(url[, new[, autoraise]])
    - Wenn für den Parameter autoraise der Wert True übergeben wird, wird versucht, das Browserfenster mit der geöffneten URL in den Vordergrund zu holen
    - Beachten Sie, dass dies bei vielen Systemen automatisch geschieht

Python Insiderwissen



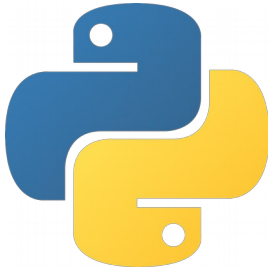
- URLs im Standardbrowser öffnen – webbrowser
  - Das Modul webbrowser dient dazu, eine Website im Standardbrowser des gerade verwendeten Systems zu öffnen
  - Die Wesentlichste Methode ist open
  - open(url[, new[, autoraise]])
    - Beispiel
      >>> import webbrowser
      >>> webbrowser.open("http://www.galileo-press.de", 2)
      True

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Das Modul functools enthält Funktionen und Dekoratoren, mit deren Hilfe sich Funktionen und Klassen auf einer abstrakten Ebene modifizieren lassen
  - Wir werden die Vereinfachung von Funktionsschnittstellen, das Hinzufügen eines Caches und das Vervollständigen der Ordnungsrelation besprechen

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen

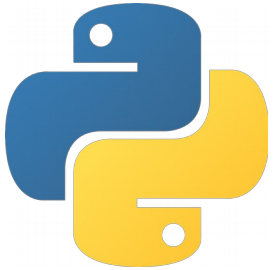
    - Im Modul functools ist die Funktion partial enthalten, mit der sich Funktionsschnittstellen vereinfachen lassen

      ```
      def f(a, b, c, d):
          print("{} {} {} {}".format(a,b,c,d))
      ```

    - Stellen Sie sich nun vor, wir müssten die Funktion f sehr häufig im Programm aufrufen und übergäben dabei für die Parameter b , c und d immer die gleichen Werte

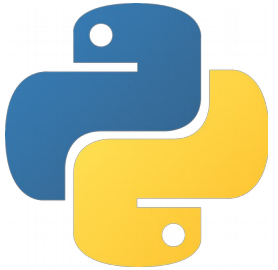
    - Es drängt sich die Frage auf, ob sich solch ein Funktionsaufruf nicht so vereinfachen lässt, dass nur der eigentlich interessante Parameter a übergeben werden muss

Python Insiderwissen



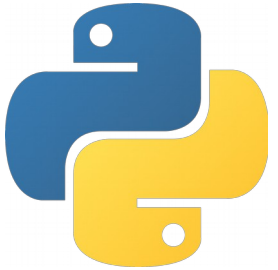
- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Die Funktion partial bekommt ein Funktionsobjekt übergeben, dessen Schnittstelle vereinfacht werden soll
      - Zusätzlich werden die zu fixierenden Positions- und Schlüsselwortparameter übergeben

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Die Funktion partial gibt ein neues Funktionsobjekt zurück, dessen Schnittstelle der von func entspricht, die jedoch um die in args (Arguments) und kwargs (Keyword Arguments) angegebenen Parameter erleichtert wurde
      - Bei einem Aufruf des zurückgegebenen Funktionsobjekts werden diese fixierten Parameter automatisch ergänzt

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen

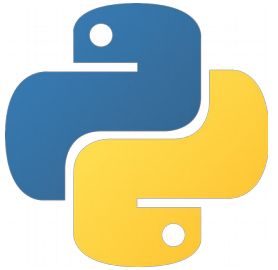
    - Und genau hier setzt die Funktion partial des Moduls functools ein

    - partial(func[, *args][, **kwargs])

      - Nutzung

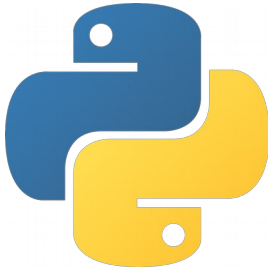
```
>>> def f(a, b, c, d):
…                 print("{} {} {} {}".format(a,b,c,d))
...
>>> import functools
>>> f_neu = functools.partial(f, b="du", c="schöne", d="Welt")
>>> f_neu("Hallo")
Hallo du schöne Welt
>>> f_neu("Tschüss")
Tschüss du schöne Welt
```

Python Insiderwissen



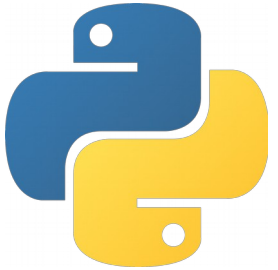
- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Zunächst wird die Funktion f definiert, die vier Parameter akzeptiert und diese hintereinander auf dem Bildschirm ausgibt
      - Da die letzten drei Parameter dieser Schnittstelle in unserem Programm immer gleich sind, möchten wir sie nicht immer wiederholen und vereinfachen die Schnittstelle mittels partial

Python Insiderwissen



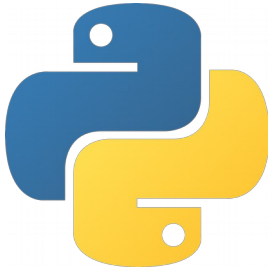
- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Dazu rufen wir die Funktion partial auf und übergeben das Funktionsobjekt von f als ersten Parameter
      - Danach folgen die drei feststehenden Werte für die Parameter b , c und d in Form von Schlüsselwortparametern
      - Die Funktion partial gibt ein Funktionsobjekt zurück, das der Funktion f mit vereinfachter Schnittstelle entspricht

Python Insiderwissen



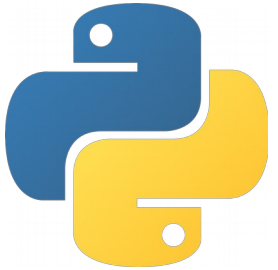
- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Dieses Funktionsobjekt kann, wie im Beispiel zu sehen, mit einem einzigen Parameter aufgerufen werden
      - Der Funktion f wird dieser Parameter gemeinsam mit den drei fixierten Parametern übergeben
      - Abgesehen von Schlüsselwortparametern können Sie der Funktion partial auch Positionsparameter übergeben, die dann ebenfalls als solche an die zu vereinfachende Funktion weitergegeben werden

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen
    - Und genau hier setzt die Funktion partial des Moduls functools ein
    - partial(func[, *args][, **kwargs])
      - Dabei ist zu beachten, dass die feststehenden Parameter dann am Anfang der Funktionsschnittstelle stehen müssen

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Funktionsschnittstellen vereinfachen

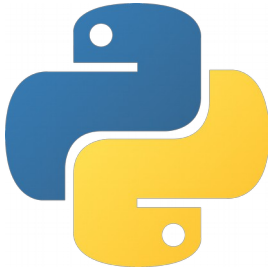
    - Und genau hier setzt die Funktion partial des Moduls functools ein

    - partial(func[, *args][, **kwargs])

      - Beispiel

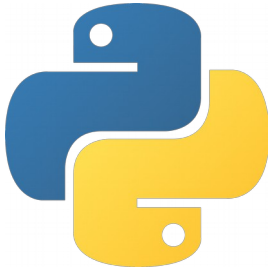
```
>>> def f(a, b, c, d):
…             print("{} {} {} {}".format(a,b,c,d))
...
>>> f_neu = functools.partial(f, "Hallo", "du", "schöne")
>>> f_neu("Welt")
Hallo du schöne Welt
>>> f_neu("Frau")
Hallo du schöne Frau
```

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - Mithilfe des Dekorators lru_cache, der im Modul functools enthalten ist, lässt sich eine Funktion mit einem Cache versehen
    - Ein Cache ist ein Speicher, der vergangene Funktionsaufrufe sichert
    - Wenn eine Parameterbelegung beim Funktionsaufruf bereits vorgekommen ist, kann das Ergebnis aus dem Cache gelesen werden, und die Funktion muss nicht noch einmal ausgeführt werden

Python Insiderwissen

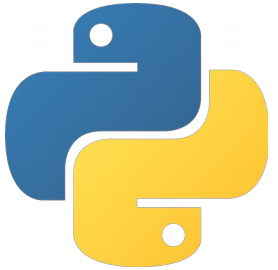


- Funktionsschnittstellen vereinfachen – functools
  - Caches

    - Dieses Prinzip kann besonders bei rechenintensiven und häufig aufgerufenen Funktionen einen großen Laufzeitvorteil bringen

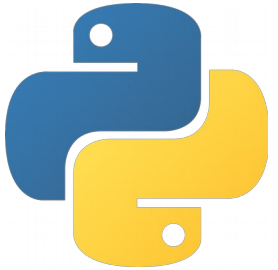
    - Hinweis
      - Wenn ein Funktionsergebnis aus dem Cache gelesen wird, wird die Funktion nicht ausgeführt
      - Das Cachen ergibt also nur Sinn, wenn die Funktion frei von Seiteneffekten und deterministisch ist, also das Ergebnis bei der gleichen Parameterbelegung stets dasselbe ist

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - @lru_cache([maxsize])
      - Der Dekorator lru_cache versieht eine Funktion mit einem LRU Cache mit maxsize Einträgen
      - Bei einem LRU Cache (für least recently used) verdrängt ein neuer Eintrag stets den am längsten nicht mehr aufgetretenen Eintrag, sofern der Cache vollständig gefüllt ist
      - Wenn für maxsize der Wert None übergeben wird, hat der Cache keine Maximalgröße und kann unbegrenzt wachsen

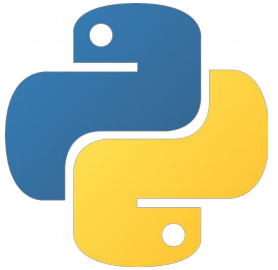
Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - @lru_cache([maxsize])
      - Beispiel
        Die Funktion fak wird zur Berechnung der Fakultät einer ganzen Zahl definiert und mit einem Cache versehen

```
>>> import functools
>>> @functools.lru_cache(20)
...     def fak(n):
…                 res = 1
…                 for i in range(2, n+1):
…                         res *= i
…                 return res
...
>>> [fak(x) for x in [7, 5, 12, 3, 5, 7, 3]]
[5040, 120, 479001600, 6, 120, 5040, 6]
```

Python Insiderwissen

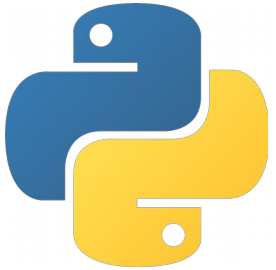


- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - @lru_cache([maxsize])
      - Mithilfe der Methode cache_info, die der Dekorator lru_cache dem dekorierten Funktionsobjekt hinzufügt, lassen sich Informationen über den aktuellen Status des Caches erhalten

        >>> fak.cache_info()
        CacheInfo(hits=3, misses=4, maxsize=20, currsize=4)

      - Das Ergebnis ist ein benanntes Tupel mit den folgenden Einträgen (nächste Folie)

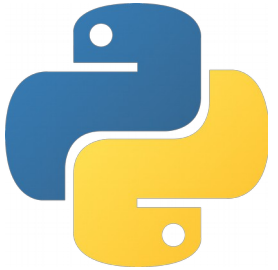
Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - @lru_cache([maxsize])

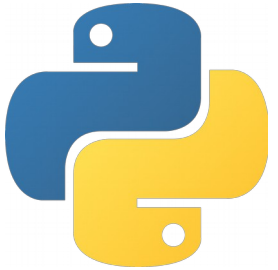
Eintrag	Beschreibung
hits	die Anzahl der Funktionsaufrufe, deren Ergebnisse aus dem Cache gelesen wurden
misses	die Anzahl der Funktionsaufrufe, deren Ergebnisse nicht aus dem Cache gelesen wurden
maxsize	die maximale Größe des Caches
currsize	die aktuelle Größe des Caches

Python Insiderwissen



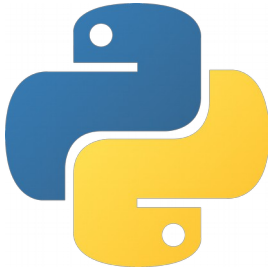
- Funktionsschnittstellen vereinfachen – functools
  - Caches
    - @lru_cache([maxsize])
      - Hinweis
        - Intern wird der Cache in Form eines Dictionarys realisiert, bei dem die Parameterbelegung eines Funktionsaufrufs als Schlüssel verwendet wird
        - Aus diesem Grund dürfen nur Instanzen von hashbaren Datentypen an ein Funktionsobjekt übergeben werden, das den hier vorgestellten LRU Cache verwendet

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Ordnungsrelationen vervollständigen
    - @lru_cache([maxsize])
      - Eine Klasse, auf der eine Ordnungsrelation definiert sein soll, für die also die Vergleichsoperatoren <, <=, >, >= funktionieren sollen, muss jede der entsprechenden magischen Methoden __lt__, __le__, __gt__ und __ge__ implementieren, obwohl eine dieser Methoden bereits ausreichen würde, um die Ordnungsrelation zu beschreiben
      - Der Dekorator total_ordering, der im Modul functools enthalten ist, erweitert eine Klasse, die nur eine der oben genannten magischen Methoden und zusätzlich die Methode __eq__ bereitstellt, um die jeweils anderen Vergleichsmethoden

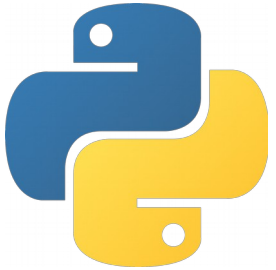
Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Ordnungsrelationen vervollständigen
    - @lru_cache([maxsize])
      - Beispiel

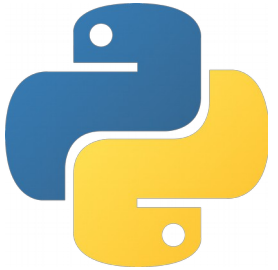
```
>>> @functools.total_ordering
... class MeinString(str):
…         def __eq__(self, other):
…             return max(self) == max(other)
...
…         def __lt__(self, other):
…             return max(self) < max(other)
...
>>> MeinString("Hallo") > MeinString("Welt")
False
>>> MeinString("Hallo") <= MeinString("Welt")
True
```

Python Insiderwissen



- Funktionsschnittstellen vereinfachen – functools
  - Ordnungsrelationen vervollständigen
    - @lru_cache([maxsize])
      - Die Klasse MeinString erbt von dem eingebauten Datentyp str
      - Instanzen von MeinString sollen anhand des größten enthaltenen Buchstabens miteinander verglichen werden
      - Dazu sind die Methoden __eq__ für den Gleichheitsoperator und __lt__ für den Kleiner-Operator implementiert
      - Da die Klasse mit total_ordering dekoriert wurde, können auch die nicht explizit implementierten Vergleichsoperatoren verwendet werden

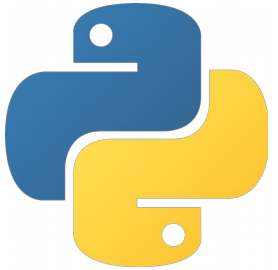
Python Insiderwissen



- Weitere Datentypen – collections
  - Das Modul collections der Standardbibliothek enthält Datentypen und Funktionen, die auf den komplexeren Basisdatentypen aufbauen und diese für bestimmte Einsatzzwecke besser nutzbar machen

Funktion bzw. Datentyp	Beschreibung
Counter	ein Dictionary zum Erfassen von Häufigkeiten
defaultdict	ein Dictionary, das einen Standardwert für nicht enthaltene Schlüssel unterstützt
deque	eine doppelt verkettete Liste
namedtuple	eine Funktion zur Erzeugung von benannten Tupeln
OrderedDict	ein Dictionary, das die Reihenfolge, in der die Schlüssel-Wert-Paare eingefügt wurden, beim Iterieren beibehält

Python Insiderwissen



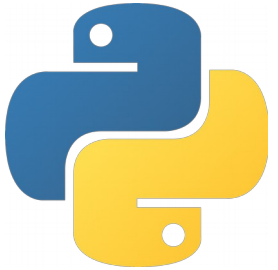
- Weitere Datentypen – collections
  - Zählen von Häufigkeiten

    - Oft ist man an der Häufigkeitsverteilung der Elemente eines iterierbaren Objekts interessiert, beispielsweise daran, wie oft einzelne Buchstaben in einem String auftreten

    - Beispiel (mit Dictionary)

```
>>> t = "Dies ist der Text"
>>> d = {}
>>> for c in t:
…         if c in d:
…             d[c] += 1
…         else:
…             d[c] = 1
...
>>> d
{' ': 3, 'e': 3, 'D': 1, 'i': 2, 'T': 1, 's': 2, 'r': 1, 't': 2, 'x': 1, 'd': 1}
```

Python Insiderwissen



- Weitere Datentypen – collections
  - Zählen von Häufigkeiten

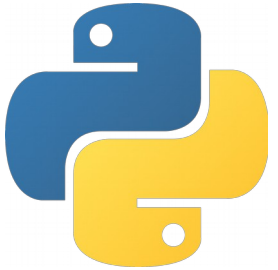
    - Der Datentyp Counter des Moduls collections ist ein Dictionary, das bei einem Zugriff mit einem unbekannten Schlüssel k automatisch ein Schlüssel-Wert-Paar k : 0 hinzufügt

    - Beispiel (mit Counter)

```
>>> t = "Dies ist der Text"
>>> d = collections.Counter()
>>> for c in t:
…         d[c] += 1
...
>>> d
Counter({' ': 3, 'e': 3, 'i': 2, 's': 2, 't': 2, 'D': 1, 'T': 1, 'r': 1,
'x': 1, 'd': 1})
```

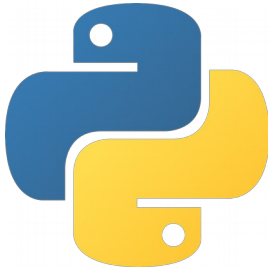
    - Im Beispiel liegen die Daten bereits in Form eines iterierbaren Objekts vor

Python Insiderwissen



- Weitere Datentypen – collections
  - Zählen von Häufigkeiten
    - In einem solchen Fall kann dieses Objekt bei der Instanziierung von Counter übergeben werden
    - Beispiel (als Einzeiler)
      >>> collections.Counter("Dies ist der Text")
      Counter({' ': 3, 'e': 3, 'i': 2, 's': 2, 't': 2, 'D': 1, 'T': 1, 'r': 1, 'x': 1, 'd': 1})
    - Alle weiteren Beispiele beziehen sich auf die Counter-Instanz d

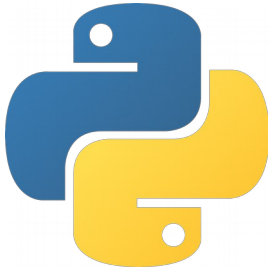
Python Insiderwissen



- Weitere Datentypen – collections
  - d.elements()
    - Diese Methode gibt einen Iterator über die Elemente einer Counter-Instanz zurück
    - Dabei wird jedes Element so oft durchlaufen, wie sein aktueller Zählerstand ist

      >>> list(d.elements())
      [' ', ' ', ' ', 'e', 'e', 'e', 'D', 'i', 'i', 'T', 's', 's', 'r', 't', 't', 'x', 'd']

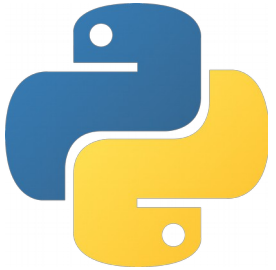
Python Insiderwissen



- Weitere Datentypen – collections
  - d.most_common([n])
    - Diese Methode gibt eine Liste der n häufigsten Elemente zurück
    - Die Liste besteht dabei aus Tupeln, die das jeweilige Element und dessen Häufigkeit enthalten

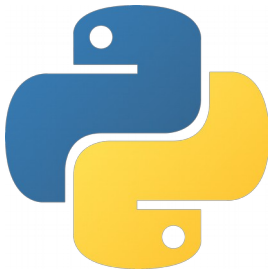
      >>> d.most_common(3)
      [(' ', 3), ('e', 3), ('i', 2)]
    - Wenn der Parameter n nicht angegeben wird, enthält die zurückgegebene Liste alle Elemente

Python Insiderwissen



- Weitere Datentypen – collections
  - d.subtract([iterable-or-mapping])
    - Diese Methode subtrahiert die Häufigkeiten der Elemente von iterable-or-mapping von den Häufigkeiten in d
    - So lassen sich beispielsweise die Zeichen finden, in deren Häufigkeiten sich deutsch- und englischsprachige Texte am besten von einander unterscheiden lassen

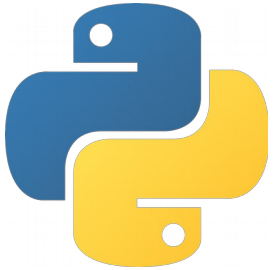
Python Insiderwissen



- Weitere Datentypen – collections
  - d.subtract([iterable-or-mapping])

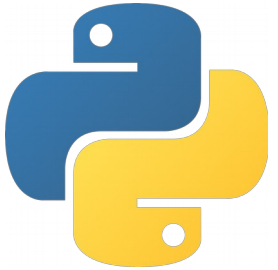
```
>>> import collections
>>> ger = collections.Counter(
…                         open("deutsch.txt", "r").read().lower())
>>> eng = collections.Counter(
…                         open("englisch.txt", "r").read().lower())
>>> eng.most_common(5)
[('e', 10890), ('a', 8567), ('i', 7879), ('n', 7676), ('t', 7530)]
>>> ger.most_common(5)
[('e', 15234), ('n', 9495), ('i', 7629), ('r', 7275), ('a', 5959)]
>>> eng.subtract(ger)
>>> eng.most_common(5)
[('o', 3633), ('a', 2608), ('t', 1734), ('y', 1504), ('p', 1238)]
```

Python Insiderwissen



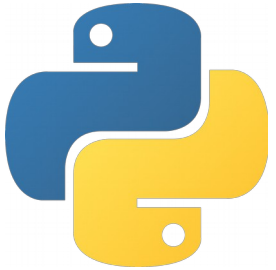
- Weitere Datentypen – collections
  - d.subtract([iterable-or-mapping])
    - Zunächst wird der Inhalt zweier gleich großer Dateien, die jeweils einen deutschen und einen englischen Text enthalten, eingelesen und eine Häufigkeitsanalyse mithilfe des Counter-Datentyps durchgeführt
    - Die Texte sind, abgesehen von Umlauten, frei von Sonderzeichen
    - Mithilfe der subtract-Methode werden die deutschen Buchstabenhäufigkeiten von den englischen abgezogen
    - Anhand des Ergebnisses lässt sich erkennen, dass sich ein englischer Text offenbar gut anhand der absoluten Häufigkeiten der Buchstaben o und a von einem deutschen unterscheiden lässt

Python Insiderwissen



- Weitere Datentypen – collections
  - d.update([iterable-or-mapping])
    - Die Funktion update verhält sich wie subtract, mit dem Unterschied, dass die in iterable-or-mapping enthaltenen Häufigkeiten nicht subtrahiert, sondern auf die Häufigkeiten von d addiert werden

Python Insiderwissen

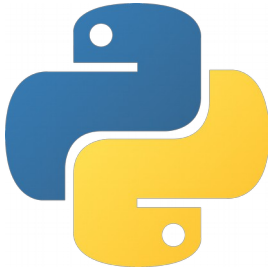


- Versteckte Passworteingaben – getpass
  - Das Modul getpass ist sehr überschaubar und ermöglicht das komfortable Einlesen eines Passworts über die Tastatur

    import getpass

  - getpass([prompt[, stream]])
    - Über den optionalen Parameter prompt kann der Text angegeben werden, der den Benutzer zur Eingabe des Passworts auffordert
    - Der Parameter ist mit "Password: " vorbelegt
    - Für den zweiten optionalen Parameter, stream, kann ein datei-ähnliches Objekt übergeben werden, in das die Aufforderung prompt geschrieben wird

Python Insiderwissen



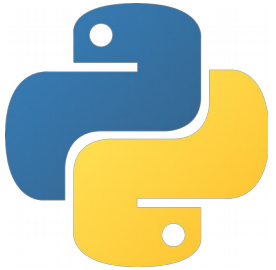
- Versteckte Passworteingaben – getpass
  - getpass([prompt[, stream]])
    - Das funktioniert nur unter Unix-ähnlichen Betriebssystemen, unter Windows wird der Parameter prompt ignoriert

      ```
      >>> s = getpass.getpass("Ihr Passwort bitte: ")
      Ihr Passwort bitte:
      >>> print(s)
      Dies ist mein Passwort
      ```

  - getpass.getuser()
    - Die Funktion getuser gibt den Namen zurück, mit dem sich der aktuelle Benutzer im Betriebssystem eingeloggt hat

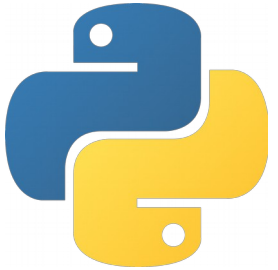
      ```
      >>> getpass.getuser()
      'Benutzername'
      ```

Python Insiderwissen



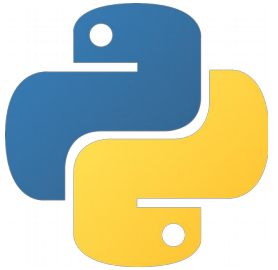
- Kommandozeilen-Interpreter – cmd
  - Das Modul cmd bietet eine einfache und abstrahierte Schnittstelle zum Schreiben eines zeilenorientierten Kommandointerpreters
  - Unter einem zeilenorientierten Kommandointerpreter versteht man eine interaktive Konsole, in der zeilenweise Kommandos eingegeben und direkt nach Bestätigung der Eingabe interpretiert werden
  - Der interaktive Modus ist ein bekanntes Beispiel für solch einen Kommandointerpreter
  - In einem eigenen Projekt ließe sich cmd beispielsweise für eine Administrator-Konsole verwenden

Python Insiderwissen



- Kommandozeilen-Interpreter – cmd
  - Das Modul cmd enthält die Klasse Cmd, die als Basisklasse für eigene Kommandointerpreter verwendet werden kann und dafür ein grobes Gerüst bereitstellt
  - Da Cmd als Basisklasse gedacht ist, ergibt es keinen Sinn, die Klasse direkt zu instanziieren
  - Das folgende Beispielprojekt verwendet die Klasse Cmd, um eine rudimentäre Konsole zu erstellen
  - Die Konsole soll die beiden Kommandos date und time verstehen und jeweils das aktuelle Datum bzw. die aktuelle Uhrzeit ausgeben
  - Beispiel – kommandozeile_cmd.py

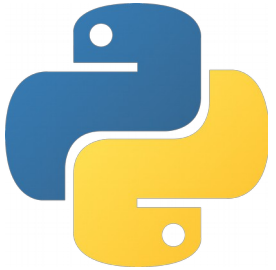
Python Insiderwissen



- Dateiinterface für Strings – io.StringIO
  - Das Modul io der Standardbibliothek enthält die Klasse StringIO, die die Schnittstelle eines Dateiobjekts bereitstellt, intern aber auf einem String arbeitet
  - Das ist beispielsweise dann nützlich, wenn eine Funktion ein geöffnetes Dateiobjekt als Parameter erwartet, um dort hineinzuschreiben, Sie die geschriebenen Daten aber lieber in Form eines Strings vorliegen haben würden
  - Hier kann in der Regel eine StringIO-Instanz übergeben werden, sodass die geschriebenen Daten danach als String weiterverwendet werden können

```
>>> from io import StringIO
>>> pseudodatei = StringIO()
```

Python Insiderwissen



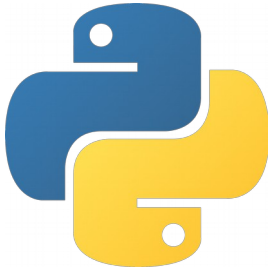
- Dateiinterface für Strings – io.StringIO
  - Dem Konstruktor kann optional ein String übergeben werden, der den anfänglichen Inhalt der Datei enthält
  - Von nun an kann die zurückgegebene Instanz, referenziert durch pseudodatei, wie ein Dateiobjekt verwendet werden

    ```
    >>> pseudodatei.write("Hallo Welt")
    >>> print(" Hallo Welt", file=pseudodatei)
    ```

  - Neben der Funktionalität eines Dateiobjekts bietet eine Instanz der Klasse StringIO eine zusätzliche Methode namens getvalue, durch die auf den internen String zugegriffen werden kann

    ```
    >>> pseudodatei.getvalue()
    'Hallo Welt Hallo Welt\n'
    ```

Python Insiderwissen



- Dateiinterface für Strings – io.StringIO
  - Wie ein Dateiobjekt auch, sollte eine StringIO -Instanz durch Aufruf der Methode close geschlossen werden, wenn sie nicht mehr gebraucht wird
  - >>> pseudodatei.close()