



合肥大學
HEFEI UNIVERSITY



Programming with Python

24. enumerate und Zwischenspiel: Pylint

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Programming with Python



Dies ist ein Kurs über das Programmieren mit der Programmiersprache Python an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist <https://thomasweise.github.io/programmingWithPython> (siehe auch den QR-Code unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielprogrammen in Python finden Sie unter <https://github.com/thomasWeise/programmingWithPythonCode>.



Outline



1. Einleitung
2. Neues Tool: Pylint
3. `enumerate`
4. Zusammenfassung





Einleitung



Einleitung

- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.



Einleitung



- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.
- Wenn wir über alle Werte `v` in `data` iterieren wollen, dann geht das mit `for v in data`.

Einleitung



- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.
- Wenn wir über alle Werte `v` in `data` iterieren wollen, dann geht das mit `for v in data`.
- Aber wenn wir auch den Index `i` der Datenelemente *während* dem iterieren wollen...
... dann geht das so nicht.

Einleitung



- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.
- Wenn wir über alle Werte `v` in `data` iterieren wollen, dann geht das mit `for v in data`.
- Aber wenn wir auch den Index `i` der Datenelemente *während* dem iterieren wollen...
...dann geht das so nicht.
- In dem Fall haben wir im Grunde zwei Möglichkeiten.

Einleitung



- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.
- Wenn wir über alle Werte `v` in `data` iterieren wollen, dann geht das mit `for v in data`.
- Aber wenn wir auch den Index `i` der Datenelemente *während* dem iterieren wollen...
...dann geht das so nicht.
- In dem Fall haben wir im Grunde zwei Möglichkeiten:
 1. Den Index anstatt der Datenvariable zum iterieren nehmen.

Einleitung



- Nehmen Sie an, das wir eine Liste `data` von Ganzzahl-Werten `v` haben.
- Wenn wir über alle Werte `v` in `data` iterieren wollen, dann geht das mit `for v in data`.
- Aber wenn wir auch den Index `i` der Datenelemente *während* dem iterieren wollen...
...dann geht das so nicht.
- In dem Fall haben wir im Grunde zwei Möglichkeiten:
 1. Den Index anstatt der Datenvariable zum iterieren nehmen.
 2. Die Datenvariable iterieren lassen, aber den Index mitzählen.

Index iterieren lassen und Container indizieren

- In dem Fall haben wir im Grunde zwei Möglichkeiten: *Den Index anstatt der Datenvariable zum iterieren nehmen...*



Index iterieren lassen und Container indizieren



- In dem Fall haben wir im Grunde zwei Möglichkeiten: *Den Index anstatt der Datenvariable zum iterieren nehmen...*
- Wir können über `data` iterieren, in dem wir die Elemente an den Indizes `i` direkt ansprechen.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Index iterieren lassen und Container indizieren



- In dem Fall haben wir im Grunde zwei Möglichkeiten: *Den Index anstatt der Datenvariable zum iterieren nehmen...*
- Wir können über `data` iterieren, in dem wir die Elemente an den Indizes `i` direkt ansprechen.
- Wir würden mit `i` über eine `range` iterieren, die von `0` bis `len(data)- 1` geht.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for stets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```


Index iterieren lassen und Container indizieren



- In dem Fall haben wir im Grunde zwei Möglichkeiten: *Den Index anstatt der Datenvariable zum iterieren nehmen...*
- Wir können über `data` iterieren, in dem wir die Elemente an den Indizes `i` direkt ansprechen.
- Wir würden mit `i` über eine `range` iterieren, die von 0 bis `len(data) - 1` geht.
- Der Kopf der Schleife wäre dann `for i in range(len(data))`.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Index iterieren lassen und Container indizieren



- In dem Fall haben wir im Grunde zwei Möglichkeiten: *Den Index anstatt der Datenvariable zum iterieren nehmen...*
- Wir können über `data` iterieren, in dem wir die Elemente an den Indizes `i` direkt ansprechen.
- Wir würden mit `i` über eine `range` iterieren, die von `0` bis `len(data) - 1` geht.
- Der Kopf der Schleife wäre dann `for i in range(len(data))`.
- Und die Datenwerte `v` bekämen wir via `data[i]`.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Index iterieren lassen und Container indizieren



- Wir können über `data` iterieren, in dem wir die Elemente an den Indizes `i` direkt ansprechen.
- Wir würden mit `i` über eine `range` iterieren, die von 0 bis `len(data) - 1` geht.
- Der Kopf der Schleife wäre dann `for i in range(len(data))`.
- Und die Datenwerte `v` bekämen wir via `data[i]`.
- Der Nachteil dieser Methode ist, dass sie *nur* für `lists` und `tuples` funktioniert.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Index iterieren lassen und Container indizieren



- Wir würden mit `i` über eine `range` iterieren, die von `0` bis `len(data) - 1` geht.
- Der Kopf der Schleife wäre dann `for i in range(len(data))`.
- Und die Datenwerte `v` bekämen wir via `data[i]`.
- Der Nachteil dieser Methode ist, dass sie *nur* für `lists` und `tuples` funktioniert.
- Nur diese beiden Kollektions-Datentypen sind direkt indizierbar.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

↓ python3 for_loop_no_enumerate_1.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen

- In dem Fall haben wir im Grunde zwei Möglichkeiten: ... *Die Datenvariable iterieren lassen, aber den Index mitzählen...*



Über Kollektion iterieren und Index manuell mitführen



- In dem Fall haben wir im Grunde zwei Möglichkeiten: ... *Die Datenvariable iterieren lassen, aber den Index mitzählen...*
- Nur `lists` und `tuples` können direkt indiziert werden.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- In dem Fall haben wir im Grunde zwei Möglichkeiten: ... *Die Datenvariable iterieren lassen, aber den Index mitzählen...*
- Nur `lists` und `tuples` können direkt indiziert werden.
- Wir können aber über alle drüber iterieren.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- In dem Fall haben wir im Grunde zwei Möglichkeiten: ... *Die Datenvariable iterieren lassen, aber den Index mitzählen...*
- Nur `lists` und `tuples` können direkt indiziert werden.
- Wir können aber über alle drüber iterieren.
- Wir könnten also „normal“ über die Kollektion drüber iterieren.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]={v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- In dem Fall haben wir im Grunde zwei Möglichkeiten: ... *Die Datenvariable iterieren lassen, aber den Index mitzählen...*
- Nur `lists` und `tuples` können direkt indiziert werden.
- Wir können aber über alle drüber iterieren.
- Wir könnten also „normal“ über die Kollektion drüber iterieren.
- Dann wissen wir natürlich nicht den Index des aktuellen Elements.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- Nur `lists` und `tuples` können direkt indiziert werden.
- Wir können aber über alle drüber iterieren.
- Wir könnten also „normal“ über die Kollektion drüber iterieren.
- Dann wissen wir natürlich nicht den Index des aktuellen Elements.
- Das macht aber nichts: Wir können einfach selbst eine Zählervariable updaten.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```


Über Kollektion iterieren und Index manuell mitführen



- Wir können aber über alle drüber iterieren.
- Wir könnten also „normal“ über die Kollektion drüber iterieren.
- Dann wissen wir natürlich nicht den Index des aktuellen Elements.
- Das macht aber nichts: Wir können einfach selbst eine Zählervariable updaten.
- Wir benutzen eine zusätzliche Ganzzahlvariable `i` und initialisieren sie mit `i: int = 0` vor der Schleif.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- Wir könnten also „normal“ über die Kollektion drüber iterieren.
- Dann wissen wir natürlich nicht den Index des aktuellen Elements.
- Das macht aber nichts: Wir können einfach selbst eine Zählervariable updaten.
- Wir benutzen eine zusätzliche Ganzzahlvariable `i` und initialisieren sie mit `i: int = 0` vor der Schleife.
- Wir erhöhen ihren Wert dann mit `i += 1` am Ende des Schleifenkörpers.

```
1  """
2  Enumerate the index-value pairs in a list: Idea 2.
3
4  We can iterate over any container class directly with a for loop.
5  In this case, we simply increment the index by ourselves.
6  """
7
8  data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- Dann wissen wir natürlich nicht den Index des aktuellen Elements.
- Das macht aber nichts: Wir können einfach selbst eine Zählervariable updaten.
- Wir benutzen eine zusätzliche Ganzzahlvariable `i` und initialisieren sie mit `i: int = 0` vor der Schleife.
- Wir erhöhen ihren Wert dann mit `i += 1` am Ende des Schleifenkörpers.
- `i += 1` ist das selbe wie `i = i + 1`.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```

Über Kollektion iterieren und Index manuell mitführen



- Das macht aber nichts: Wir können einfach selbst eine Zählervariable updaten.
- Wir benutzen eine zusätzliche Ganzzahlvariable `i` und initialisieren sie mit `i: int = 0` vor der Schleife.
- Wir erhöhen ihren Wert dann mit `i += 1` am Ende des Schleifenkörpers.
- `i += 1` ist das selbe wie `i = i + 1`.
- Diese Methode funktioniert bei allen Kollektionsdatentypen.

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

↓ python3 for_loop_no_enumerate_2.py ↓

```
1 data[0]=1
2 data[1]=2
3 data[2]=3
4 data[3]=5
```



Neues Tool: Pylint



Pylint



- Pylint ist ein Python Linter der Code nach Stilinkonsistenzen, möglichen Fehlern, und möglichen Verbesserungen durchsucht¹⁸

Pylint



- Pylint ist ein Python Linter der Code nach Stilinkonsistenzen, möglichen Fehlern, und möglichen Verbesserungen durchsucht¹⁸
- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.

```
twaise@weise-laptop: ~  
twaise@weise-laptop:~$ pip install pylint  
Collecting pylint  
  Using cached pylint-3.3.0-py3-none-any.whl.metadata (12 kB)  
Collecting platformdirs>=2.2.0 (from pylint)  
  Using cached platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB)  
Collecting astroid<=3.4.0-dev0,>=3.3.3 (from pylint)  
  Using cached astroid-3.3.4-py3-none-any.whl.metadata (4.5 kB)  
Collecting isort!=5.13.0,<6,>=4.2.5 (from pylint)  
  Using cached isort-5.13.2-py3-none-any.whl.metadata (12 kB)  
Collecting mccabe<0.8,>=0.6 (from pylint)  
  Using cached mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)  
Collecting tomlkit>=0.10.1 (from pylint)  
  Using cached tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)  
Collecting dill>=0.3.6 (from pylint)  
  Using cached dill-0.3.8-py3-none-any.whl.metadata (10 kB)  
Using cached pylint-3.3.0-py3-none-any.whl (521 kB)  
Using cached astroid-3.3.4-py3-none-any.whl (274 kB)  
Using cached dill-0.3.8-py3-none-any.whl (116 kB)  
Using cached isort-5.13.2-py3-none-any.whl (92 kB)  
Using cached mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)  
Using cached platformdirs-4.3.6-py3-none-any.whl (18 kB)  
Using cached tomlkit-0.13.2-py3-none-any.whl (37 kB)  
Installing collected packages: tomlkit, platformdirs, mccabe, isort, dill, astro  
id, pylint  
Successfully installed astroid-3.3.4 dill-0.3.8 isort-5.13.2 mccabe-0.7.0 platfo  
rmdirs-4.3.6 pylint-3.3.0 tomlkit-0.13.2  
twaise@weise-laptop:~$
```

Pylint



- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.
- Sie würden dann `pip install pylint` eintippen und `↵` drücken.

```
tweise@weise-laptop: ~$ pip install pylint
Collecting pylint
  Using cached pylint-3.3.0-py3-none-any.whl.metadata (12 kB)
Collecting platformdirs>=2.2.0 (from pylint)
  Using cached platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB)
Collecting astroid<=3.4.0-dev0,>=3.3.3 (from pylint)
  Using cached astroid-3.3.4-py3-none-any.whl.metadata (4.5 kB)
Collecting isort!=5.13.0,<6,>=4.2.5 (from pylint)
  Using cached isort-5.13.2-py3-none-any.whl.metadata (12 kB)
Collecting mccabe<0.8,>=0.6 (from pylint)
  Using cached mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting tomlkit>=0.10.1 (from pylint)
  Using cached tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Collecting dill>=0.3.6 (from pylint)
  Using cached dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Using cached pylint-3.3.0-py3-none-any.whl (521 kB)
Using cached astroid-3.3.4-py3-none-any.whl (274 kB)
Using cached dill-0.3.8-py3-none-any.whl (116 kB)
Using cached isort-5.13.2-py3-none-any.whl (92 kB)
Using cached mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Using cached platformdirs-4.3.6-py3-none-any.whl (18 kB)
Using cached tomlkit-0.13.2-py3-none-any.whl (37 kB)
Installing collected packages: tomlkit, platformdirs, mccabe, isort, dill, astroid, pylint
Successfully installed astroid-3.3.4 dill-0.3.8 isort-5.13.2 mccabe-0.7.0 platformdirs-4.3.6 pylint-3.3.0 tomlkit-0.13.2
tweise@weise-laptop: ~$
```

Pylint



- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.
- Sie würden dann `pip install pylint` eintippen und `↵` drücken.
- Normalerweise machen Sie dass unter einem virtuellen Environment, was wir später diskutieren.

```
tweise@weise-laptop: ~$ pip install pylint
Collecting pylint
  Using cached pylint-3.3.0-py3-none-any.whl.metadata (12 kB)
Collecting platformdirs>=2.2.0 (from pylint)
  Using cached platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB)
Collecting astroid<=3.4.0-dev0,>=3.3.3 (from pylint)
  Using cached astroid-3.3.4-py3-none-any.whl.metadata (4.5 kB)
Collecting isort!=5.13.0,<6,>=4.2.5 (from pylint)
  Using cached isort-5.13.2-py3-none-any.whl.metadata (12 kB)
Collecting mccabe<0.8,>=0.6 (from pylint)
  Using cached mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting tomlkit>=0.10.1 (from pylint)
  Using cached tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Collecting dill>=0.3.6 (from pylint)
  Using cached dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Using cached pylint-3.3.0-py3-none-any.whl (521 kB)
Using cached astroid-3.3.4-py3-none-any.whl (274 kB)
Using cached dill-0.3.8-py3-none-any.whl (116 kB)
Using cached isort-5.13.2-py3-none-any.whl (92 kB)
Using cached mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Using cached platformdirs-4.3.6-py3-none-any.whl (18 kB)
Using cached tomlkit-0.13.2-py3-none-any.whl (37 kB)
Installing collected packages: tomlkit, platformdirs, mccabe, isort, dill, astro
id, pylint
Successfully installed astroid-3.3.4 dill-0.3.8 isort-5.13.2 mccabe-0.7.0 platfo
rmdirs-4.3.6 pylint-3.3.0 tomlkit-0.13.2
tweise@weise-laptop: ~$
```

Pylint



- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.
- Sie würden dann `pip install pylint` eintippen und `↵` drücken.
- Normalerweise machen Sie dass unter einem virtuellen Environment, was wir später diskutieren.
- So oder so, Pylint wird installiert.

```
tweise@weise-laptop: ~$ pip install pylint
Collecting pylint
  Using cached pylint-3.3.0-py3-none-any.whl.metadata (12 kB)
Collecting platformdirs>=2.2.0 (from pylint)
  Using cached platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB)
Collecting astroid<=3.4.0-dev0,>=3.3.3 (from pylint)
  Using cached astroid-3.3.4-py3-none-any.whl.metadata (4.5 kB)
Collecting isort!=5.13.0,<6,>=4.2.5 (from pylint)
  Using cached isort-5.13.2-py3-none-any.whl.metadata (12 kB)
Collecting mccabe<0.8,>=0.6 (from pylint)
  Using cached mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB)
Collecting tomlkit>=0.10.1 (from pylint)
  Using cached tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Collecting dill>=0.3.6 (from pylint)
  Using cached dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Using cached pylint-3.3.0-py3-none-any.whl (521 kB)
Using cached astroid-3.3.4-py3-none-any.whl (274 kB)
Using cached dill-0.3.8-py3-none-any.whl (116 kB)
Using cached isort-5.13.2-py3-none-any.whl (92 kB)
Using cached mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Using cached platformdirs-4.3.6-py3-none-any.whl (18 kB)
Using cached tomlkit-0.13.2-py3-none-any.whl (37 kB)
Installing collected packages: tomlkit, platformdirs, mccabe, isort, dill, astroid, pylint
Successfully installed astroid-3.3.4 dill-0.3.8 isort-5.13.2 mccabe-0.7.0 platformdirs-4.3.6 pylint-3.3.0 tomlkit-0.13.2
twaise@weise-laptop: ~$
```


Pylint



- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.
- Sie würden dann `pip install pylint` eintippen und `↵` drücken.
- Normalerweise machen Sie dass unter einem virtuellen Environment, was wir später diskutieren.
- So oder so, Pylint wird installiert.
- Wenn wir Pylint auf eine Datei `fileToScan.py` anwenden wollen, dann tippen wir `pylint fileToScan.py` in ein Terminal (wobei `fileToScan.py` auch ein Verzeichnis seien).

Pylint



- Um dieses Programm zu installieren, öffnen Sie ein Terminal, in dem Sie unter Ubuntu `Ctrl` + `Alt` + `T` drücken und unter Microsoft Windows mit Druck auf `Windows` + `R`, dann Schreiben von `cmd`, dann Druck auf `↵`.
- Sie würden dann `pip install pylint` eintippen und `↵` drücken.
- Normalerweise machen Sie dass unter einem virtuellen Environment, was wir später diskutieren.
- So oder so, Pylint wird installiert.
- Wenn wir Pylint auf eine Datei `fileToScan.py` anwenden wollen, dann tippen wir `pylint fileToScan.py` in ein Terminal (wobei `fileToScan.py` auch ein Verzeichnis seien).
- Dabei können wir einige Linter-Regeln mit dem Parameter `--disable=...` deaktivieren, wie wir es auch bei Ruff machen.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.
- Die erste funktioniert nur bei Listen und Tupeln.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.
- Die erste funktioniert nur bei Listen und Tupeln.
- Die zweite zwingt uns, eine Index-Variable manuell zu initialisieren und hochzuzählen.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.
- Die erste funktioniert nur bei Listen und Tupeln.
- Die zweite zwingt uns, eine Index-Variable manuell zu initialisieren und hochzuzählen.
- Was sagen Pylint und Ruff zu diesen Ideen?

Was sagen Pylint und Ruff dazu?



```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):    # We iterate over indices and
12     print(f"data[{i}]= {data[i]}") # use them to get the data elements.
```

Was sagen Pylint und Ruff dazu?



```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]={data[i]}") # use them to get the data elements.
```

```
1 $ pylint for_loop_no_enumerate_1.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2 ***** Module for_loop_no_enumerate_1
3 for_loop_no_enumerate_1.py:11:0: C0200: Consider using enumerate instead
   ↳ of iterating with range and len (consider-using-enumerate)
4
5 -----
6 Your code has been rated at 6.67/10
7
8 # pylint 4.0.2 failed with exit code 16.
```


Was sagen Pylint und Ruff dazu?



```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]= {data[i]}") # use them to get the data elements.
```

```
1 $ pylint for_loop_no_enumerate_1.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2 ***** Module for_loop_no_enumerate_1
3 for_loop_no_enumerate_1.py:11:0: C0200: Consider using enumerate instead
   ↳ of iterating with range and len (consider-using-enumerate)
4
5 -----
6 Your code has been rated at 6.67/10
7
8 # pylint 4.0.2 failed with exit code 16.
```

```
1 $ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
   ↳ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
   ↳ NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
   ↳ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
   ↳ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
   ↳ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
   ↳ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
   ↳ T201,TRY003,UP035,W --line-length 79 for_loop_no_enumerate_1.py
2 All checks passed!
3 # ruff 0.14.2 succeeded with exit code 0.
```


Was sagen Pylint und Ruff dazu?

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]= {data[i]}") # use them to get the data elements.
```

```
1 $ pylint for_loop_no_enumerate_1.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2 ***** Module for_loop_no_enumerate_1
3 for_loop_no_enumerate_1.py:11:0: C0200: Consider using enumerate instead
   ↳ of iterating with range and len (consider-using-enumerate)
4
5 -----
6 Your code has been rated at 6.67/10
7
8 # pylint 4.0.2 failed with exit code 16.
```

```
1 $ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
   ↳ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
   ↳ NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
   ↳ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
   ↳ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
   ↳ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
   ↳ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
   ↳ T201,TRY003,UP035,W --line-length 79 for_loop_no_enumerate_1.py
2 All checks passed!
3 # ruff 0.14.2 succeeded with exit code 0.
```

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

Was sagen Pylint und Ruff dazu?

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]= {data[i]}") # use them to get the data elements.
```

```
1 $ pylint for_loop_no_enumerate_1.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2 ***** Module for_loop_no_enumerate_1
3 for_loop_no_enumerate_1.py:11:0: C0200: Consider using enumerate instead
   ↳ of iterating with range and len (consider-using-enumerate)
4
5 -----
6 Your code has been rated at 6.67/10
7
8 # pylint 4.0.2 failed with exit code 16.
```

```
1 $ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
   ↳ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
   ↳ NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
   ↳ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
   ↳ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
   ↳ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
   ↳ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
   ↳ T201,TRY003,UP035,W --line-length 79 for_loop_no_enumerate_1.py
2 All checks passed!
3 # ruff 0.14.2 succeeded with exit code 0.
```

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]= {v}")
13     i += 1 # And we increment the index by ourselves.
```

```
1 $ pylint for_loop_no_enumerate_2.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2
3 -----
4 Your code has been rated at 10.00/10
5
6 # pylint 4.0.2 succeeded with exit code 0.
```

Was sagen Pylint und Ruff dazu?

```
1 """
2 Enumerate the index-value pairs in a list: Idea 1.
3
4 Lists and tuples can be indexed directly. Therefore, we can let the
5 index `i` go from `0` to `len(data)-1` and access the data elements by
6 using the index `i`. This would not work for sets or dicts.
7 """
8
9 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
10
11 for i in range(len(data)):      # We iterate over indices and
12     print(f"data[{i}]=data[i]") # use them to get the data elements.
```

```
1 $ pylint for_loop_no_enumerate_1.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
2 ***** Module for_loop_no_enumerate_1
3 for_loop_no_enumerate_1.py:11:0: C0200: Consider using enumerate instead
   ↳ of iterating with range and len (consider-using-enumerate)
4
5 -----
6 Your code has been rated at 6.67/10
7
8 # pylint 4.0.2 failed with exit code 16.
```

```
1 $ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
   ↳ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
   ↳ NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
   ↳ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
   ↳ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
   ↳ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
   ↳ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
   ↳ T201,TRY003,UP035,W --line-length 79 for_loop_no_enumerate_1.py
2 All checks passed!
3 # ruff 0.14.2 succeeded with exit code 0.
```

```
1 """
2 Enumerate the index-value pairs in a list: Idea 2.
3
4 We can iterate over any container class directly with a for loop.
5 In this case, we simply increment the index by ourselves.
6 """
7
8 data: list[int] = [1, 2, 3, 5] # The data to iterate over.
9
10 i = 0 # The initial index is 0.
11 for v in data: # We iterate over the data values directly.
12     print(f"data[{i}]=data[i]")
13     i += 1 # And we increment the index by ourselves.
```

```
1 $ pylint for_loop_no_enumerate_2.py --disable=C0103,C0302,C0325,R0801,
   ↳ R0901,R0902,R0903,R0911,R0912,R0913,R0914,R0915,R1702,R1728,W0212,
   ↳ W0238,W0703
```

```
2 -----
3 Your code has been rated at 10.00/10
4
5 # pylint 4.0.2 succeeded with exit code 0.
```

```
1 $ ruff check --target-version py312 --select=A,AIR,ANN,ASYNC,B,BLE,C,C4,
   ↳ COM,D,DJ,DTZ,E,ERA,EXE,F,FA,FIX,FLY,FURB,G,I,ICN,INP,ISC,INT,LOG,N
   ↳ NPY,PERF,PIE,PLC,PLE,PLR,PLW,PT,PYI,Q,RET,RSE,RUF,S,SIM,T,T10,TD,
   ↳ TID,TRY,UP,W,YTT --ignore=A005,ANN001,ANN002,ANN003,ANN204,ANN401,
   ↳ B008,B009,B010,C901,D203,D208,D212,D401,D407,D413,INP001,N801,
   ↳ PLC2801,PLR0904,PLR0911,PLR0912,PLR0913,PLR0914,PLR0915,PLR0916,
   ↳ PLR0917,PLR1702,PLR2004,PLR6301,PT011,PT012,PT013,PYI041,RUF100,S,
   ↳ T201,TRY003,UP035,W --line-length 79 for_loop_no_enumerate_2.py
2 SIM113 Use `enumerate()` for index variable `i` in `for` loop
3 --> for_loop_no_enumerate_2.py:13:5
4 |
5 | for v in data: # We iterate over the data values directly.
6 |     print(f"data[{i}]=data[i]")
7 |     i += 1 # And we increment the index by ourselves.
8 |     .....
9 |
10 |
11 Found 1 error.
12 # ruff 0.14.2 failed with exit code 1.
```

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.
- Die erste funktioniert nur bei Listen und Tupeln.
- Die zweite zwingt uns, eine Index-Variable manuell zu initialisieren und hochzuzählen.
- Was sagen Pylint und Ruff zu diesen Ideen?
- OK. Pylint mag die erste Variante nicht, Ruff kann die zweite nicht leiden.

Was sagen Pylint und Ruff dazu?



- Wir haben zwei verschiedene Ansätze entwickelt, um über die **Daten** in einer Kollektion zu iterieren und *gleichzeitig* den aktuellen **Index** zu kennen.
- Beide Ideen sind etwas eigenartig.
- Die erste funktioniert nur bei Listen und Tupeln.
- Die zweite zwingt uns, eine Index-Variable manuell zu initialisieren und hochzuzählen.
- Was sagen Pylint und Ruff zu diesen Ideen?
- OK. Pylint mag die erste Variante nicht, Ruff kann die zweite nicht leiden.
- In beiden Fällen schlagen die Linter vor, stattdessen eine Funktion `enumerate` zu nehmen.

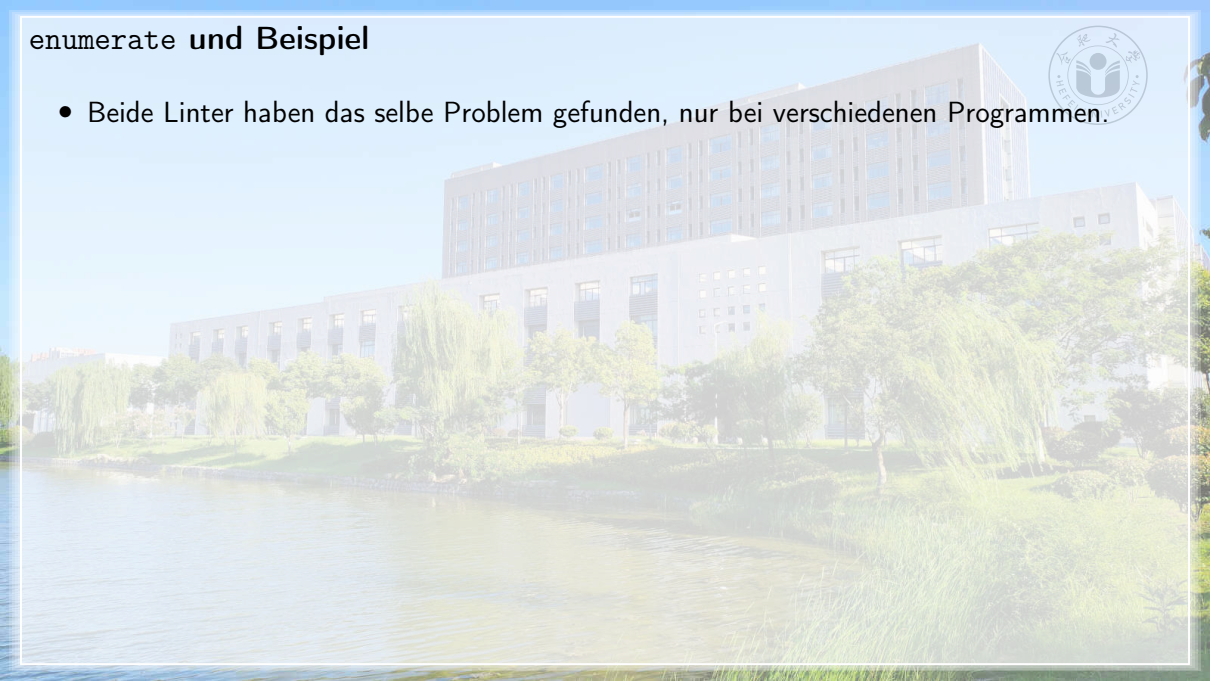


enumerate



enumerate und Beispiel

- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.



enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?
- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?
- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?
- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.
- Darum können wir den Kopf unserer Schleife zu `for i, v in enumerate(data)` umbauen.

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?
- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.
- Darum können wir den Kopf unserer Schleife zu `for i, v in enumerate(data)` umbauen.
- Beachten Sie, dass hier der Index `i` zuerst kommt und der Datenwert `v` an zweiter Stelle.

enumerate und Beispiel



- Beide Linter haben das selbe Problem gefunden, nur bei verschiedenen Programmen.
- Beide haben vorgeschlagen, eine Funktion `enumerate` zu benutzen, um über eine Kollektion zu iterieren und gleichzeitig den Index des aktuellen Elements zu kennen.
- Was macht `enumerate`?
- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.
- Darum können wir den Kopf unserer Schleife zu `for i, v in enumerate(data)` umbauen.
- Beachten Sie, dass hier der Index `i` zuerst kommt und der Datenwert `v` an zweiter Stelle.
- Schauen wir uns das mal an.

enumerate und Beispiel



- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.

```
1  """
2  Enumerate the index-value pairs in a list using enumerate.
3
4  We now use the function `enumerate` to generate a sequence of
5  index-value pairs and directly unpack them in the loop header into the
6  variables `i` (for the index) and `v` (for the value).
7  """
8
9  data: list[int] = [1, 2, 3, 5]  # The data to iterate over.
10
11  for i, v in enumerate(data):    # Generate index-value pair sequence...
12      print(f"data[{i}]= {v}")
```

↓ python3 for_loop_enumerate.py ↓

```
1  data[0]=1
2  data[1]=2
3  data[2]=3
4  data[3]=5
```


enumerate und Beispiel



- `enumerate` akzeptiert eine Kollektion als Parameter und generiert eine Sequenz von Index-Wert Tupeln⁸.
- Wir wissen ja schon, dass wir solche Tupel beim Iterieren über Sequenzen gleich entpacken können.
- Darum können wir den Kopf unserer Schleife zu `for i, v in enumerate(data)` umbauen.
- Beachten Sie, dass hier der Index `i` zuerst kommt und der Datenwert `v` an zweiter Stelle.
- Schauen wir uns das mal an.
- Das würde mit allen Kollektionen genau gleich funktionieren, egal ob `list`, `set`, `tuple`, oder `dictionary`.



Zusammenfassung



Zusammenfassung



- Wir haben zwei nützliche Dinge gelernt.

Zusammenfassung



- Wir haben zwei nützliche Dinge gelernt:
 1. Jetzt kennen wir zwei Linter-Programme, nämlich Pylint und Ruff.

Zusammenfassung



- Wir haben zwei nützliche Dinge gelernt:
 1. Jetzt kennen wir zwei Linter-Programme, nämlich Pylint und Ruff.
 2. Mit `enumerate` kennen wir ein Methode, über beliebige Sequenzen iterieren zu können und dabei immer den Index des aktuellen Elements zu kennen.

Zusammenfassung



- Wir haben zwei nützliche Dinge gelernt:
 1. Jetzt kennen wir zwei Linter-Programme, nämlich Pylint und Ruff.
 2. Mit `enumerate` kennen wir eine Methode, über beliebige Sequenzen iterieren zu können und dabei immer den Index des aktuellen Elements zu kennen.
- Sehr gut.



谢谢你们！
Thank you!
Vielen Dank!



References I



- [1] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 69, 70).
- [2] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 69).
- [3] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 69).
- [4] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 70).
- [5] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0224-9 (siehe S. 69).
- [6] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 69).
- [7] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 70).
- [8] Raymond Hettinger. *The `enumerate()` Built-In Function*. Python Enhancement Proposal (PEP) 279. Beaverton, OR, USA: Python Software Foundation (PSF), 30. Jan. 2002. URL: <https://peps.python.org/pep-0279> (besucht am 2025-09-02) (siehe S. 50–59).
- [9] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 69).
- [10] Stephen Curtis Johnson. *Lint, a C Program Checker*. Computing Science Technical Report 78–1273. New York, NY, USA: Bell Telephone Laboratories, Incorporated, 25. Okt. 1978. URL: <https://wolfram.schneider.org/bsd/7thEdManVol2/lint/lint.pdf> (besucht am 2024-08-23) (siehe S. 69).
- [11] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 69).

References II



- [12] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 69).
- [13] Charlie Marsh. "Ruff". In: URL: <https://pypi.org/project/ruff> (besucht am 2025-08-29) (siehe S. 69).
- [14] Charlie Marsh. *ruff: An Extremely Fast Python Linter and Code Formatter, Written in Rust*. New York, NY, USA: Astral Software Inc., 28. Aug. 2022. URL: <https://docs.astral.sh/ruff> (besucht am 2024-08-23) (siehe S. 69).
- [15] Carl Meyer. *Python Virtual Environments*. Python Enhancement Proposal (PEP) 405. Beaverton, OR, USA: Python Software Foundation (PSF), 13. Juni 2011–24. Mai 2012. URL: <https://peps.python.org/pep-0405> (besucht am 2024-12-25) (siehe S. 70).
- [16] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 69).
- [17] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/31EC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Nov. 2017. URL: <https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf> (besucht am 2024-06-29) (siehe S. 69).
- [18] Pylint Contributors. *Pylint*. Toulouse, Occitanie, France: Logilab, 2003–2024. URL: <https://pylint.readthedocs.io/en/stable> (besucht am 2024-09-24) (siehe S. 29, 30, 69).
- [19] Yeonhee Ryou, Sangwoo Joh, Joonmo Yang, Sujin Kim und Youil Kim. "Code Understanding Linter to Detect Variable Misuse". In: *37th IEEE/ACM International Conference on Automated Software Engineering (ASE'2022)*. 10.–14. Okt. 2022, Rochester, MI, USA. New York, NY, USA: Association for Computing Machinery (ACM), 2022, 133:1–133:5. ISBN: 978-1-4503-9475-8. doi:10.1145/3551349.3559497 (siehe S. 69).
- [20] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 69).
- [21] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 69).

References III



- [22] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: **978-0-13-792931-3** (siehe S. 69).
- [23] "Virtual Environments and Packages". In: *Python 3 Documentation. The Python Tutorial*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. Kap. 12. URL: <https://docs.python.org/3/tutorial/venv.html> (besucht am 2024-12-24) (siehe S. 70).
- [24] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: <https://thomasweise.github.io/programmingWithPython> (besucht am 2025-01-05) (siehe S. 69).
- [25] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: **978-1-78439-687-9** (siehe S. 69).

Glossary (in English) I



Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs^{3,16,25}. Learn more at <https://www.gnu.org/software/bash>.

C is a programming language, which is very successful in system programming situations^{5,17}.

IT information technology

linter A linter is a tool for analyzing program code to identify bugs, problems, vulnerabilities, and inconsistent code styles^{10,19}. Ruff is an example for a linter used in the Python world.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{1,6,20–22}. We recommend using it for this course, for software development, and for research. Learn more at <https://www.linux.org>. Its variant Ubuntu is particularly easy to use and install.

Microsoft Windows is a commercial proprietary operating system². It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at <https://www.microsoft.com/windows>.






Pylint is a linter for Python that checks for errors, enforces coding standards, and that can make suggestions for improvements¹⁸. Learn more at <https://www.pylint.org>.

Python The Python programming language^{9,11,12,24}, i.e., what you will learn about in our book²⁴. Learn more at <https://python.org>.

Ruff is a linter and code formatting tool for Python^{13,14}. Learn more at <https://docs.astral.sh/ruff> or in²⁴.

Glossary (in English) II



terminal A terminal is a text-based window where you can enter commands and execute them^{1,4}. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf  + , dann Schreiben von `cmd`, dann Druck auf . Under Ubuntu Linux,  +  opens a terminal, which then runs a Bash shell inside.

Ubuntu is a variant of the open source operating system Linux^{4,7}. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at <https://ubuntu.com>. If you are in China, you can download it from <https://mirrors.ustc.edu.cn/ubuntu-releases>.

virtual environment A virtual environment is a directory that contains a local Python installation^{15,23}. It comes with its own package installation directory. Multiple different virtual environments can be installed on a system. This allows different applications to use different versions of the same packages without conflict, because we can simply install these applications into different virtual environments.