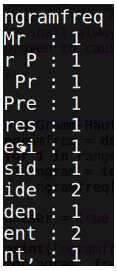
```
# Länge der verwendeten N-Gramme
L = int(sys.argv[1])

# Trainingstext aus Datei einlesen
with open(sys.argv[2]) as file:
    text = file.read()
```

text = “Mr President, I believe that the precautionary principle needs to be quite radical in order to work, otherwise we shall always be confused by all the different interpretations possible. I will give an example: a fungicide has been proven to cause babies to be born blind. It is therefore a teratogenic substance.”

```
# N-Gramm-Häufigkeiten berechnen
ngramfreq = defaultdict(int)
for i in range(len(text) - L+1):
    ngram = text[i:i+L]
    ngramfreq[ngram] += 1
```



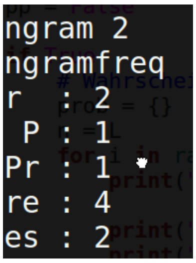
```
ngramfreq
Mr  : 1
r P : 1
 Pr : 1
Pre : 1
res : 1
esi : 1
sid : 1
ide : 2
den : 1
ent : 2
nt, : 1
```

$$p(w_i|w_{i-k}^{i-1}) = \frac{\max(0, f(w_{i-k}^{i}) - \delta_k)}{f(w_{i-k}^{i-1})} + \alpha(w_{i-k}^{i-1})\, p(w_i|w_{i-k\underline{+1}}^{i-1})$$

```
for _ in range(L):

    # Berechnung des Discounts
    N1 = sum(1 for v in ngramfreq.values() if v == 1)
    N2 = sum(1 for v in ngramfreq.values() if v == 2)

    if N1 > 100:
        discount = N1 / (N1 + 2.0 * N2)
    else:
        # Defaultwert für den Discount verwenden, falls die Zahl
        # der einmal aufgetretenen N-Gramme zu klein ist.
        discount = 0.5

    # Berechnung aller Kontexthäufigkeiten
    contextfreq = defaultdict(int)
    for ngram, freq in ngramfreq.items():
        context = ngram[:-1]
        contextfreq[context] += freq

    # Berechnung der relativen Häufigkeiten mit Discount
    for ngram, f in ngramfreq.items():
        context = ngram[:-1]
        prob[ngram] = (ngramfreq[ngram] - discount) / contextfreq[context]
```

$$\delta = \frac{N_1}{N_1 + 2N_2}$$

```
# Berechnung der N-1Gramm-Häufigkeiten
sngramfreq = defaultdict(int)
for ngram, freq in ngramfreq.items():
    sngramfreq[ngram[1:]] += freq
ngramfreq = sngramfreq
```

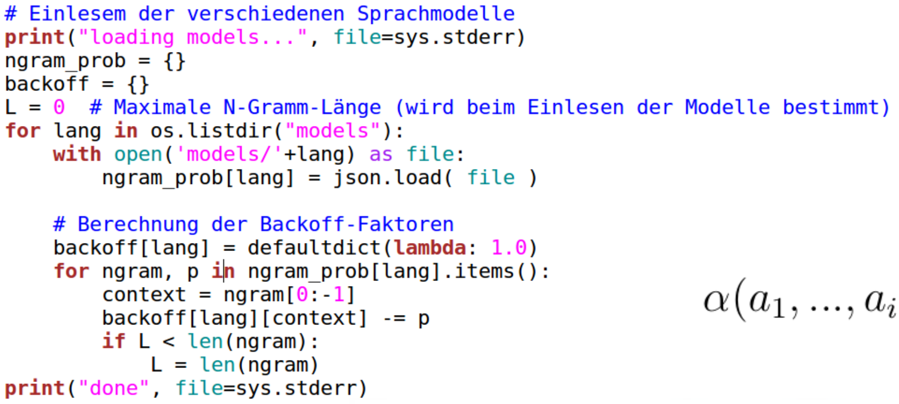


```
ngram 2
ngramfreq
r   : 2
 P  : 1
Pr  : 1
re  : 4
es  : 2
```

```
# Parameter in Datei speichern
with open(sys.argv[3],"wt") as file:
    json.dump(prob, file)
```

Prob is die Relative Häufigkeiten mit Discount.
Wir berechnen das für alle Sprache

guess_language.py (read p* dictionary of all languages, then read the input text, extract ngram of the text, compute p_smoothed(ngram) for each ngram, multiply them to get a score, then make prediction about the language of the text)

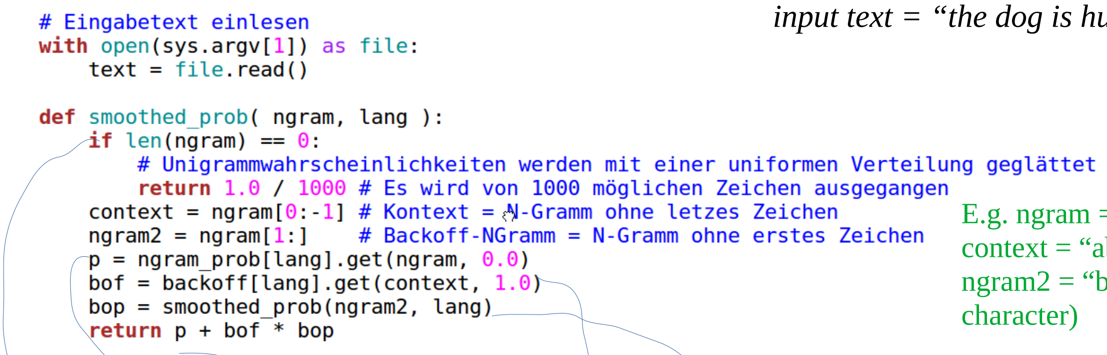


```
# Einlesem der verschiedenen Sprachmodelle
print("loading models...", file=sys.stderr)
ngram_prob = {}
backoff = {}
L = 0  # Maximale N-Gramm-Länge (wird beim Einlesen der Modelle bestimmt)
for lang in os.listdir("models"):
    with open('models/'+lang) as file:
        ngram_prob[lang] = json.load( file )

    # Berechnung der Backoff-Faktoren
    backoff[lang] = defaultdict(lambda: 1.0)
    for ngram, p in ngram_prob[lang].items():
        context = ngram[0:-1]
        backoff[lang][context] -= p
        if L < len(ngram):
            L = len(ngram)
print("done", file=sys.stderr)
```

$$\alpha(a_1, ..., a_i) = 1 - \sum_a p^*(a|a_1, ..., a_i)$$

backoff = { “en”: {“ab”: 0.003 , “cd”: 0.1 , …} ,
“de” : {“ab”: 0.015 , “cd”: 0.1 , …}
}

input text = “the dog is hungry”



```
# Eingabetext einlesen
with open(sys.argv[1]) as file:
    text = file.read()

def smoothed_prob( ngram, lang ):
    if len(ngram) == 0:
        # Unigrammwahrscheinlichkeiten werden mit einer uniformen Verteilung geglättet
        return 1.0 / 1000 # Es wird von 1000 möglichen Zeichen ausgegangen
    context = ngram[0:-1] # Kontext = N-Gramm ohne letzes Zeichen
    ngram2 = ngram[1:]    # Backoff-NGramm = N-Gramm ohne erstes Zeichen
    p = ngram_prob[lang].get(ngram, 0.0)
    bof = backoff[lang].get(context, 1.0)
    bop = smoothed_prob(ngram2, lang)
    return p + bof * bop
```

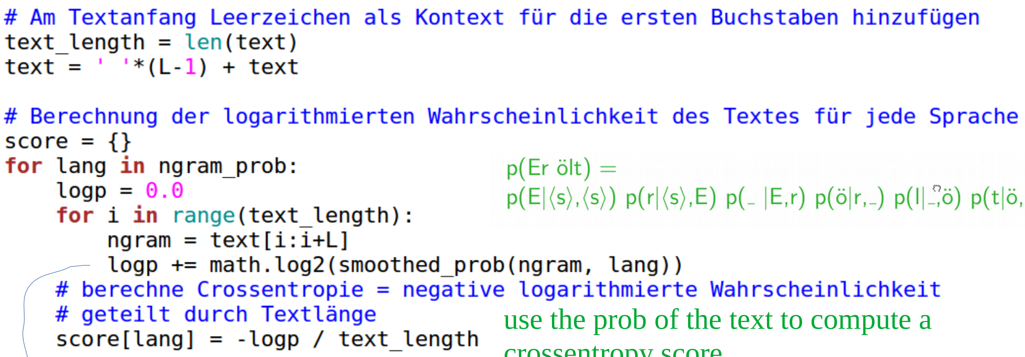
E.g. ngram = “abc”
context = “ab”
ngram2 = “bc” (after removing one context character)

$$p_L(a_i|a_1^{i-1}) = p_L^*(a_i|a_1^{i-1}) + \alpha(a_1^{i-1})p_L(a_i|a_2^{i-1})$$

$$p(a_1) = p_L^*(a_1) + \alpha()\frac{1}{1000}$$

This function computes the smoothed_prob of a given ngram of a specific language using p* and backoff information.
The the reduced ngram part (backoff part), it passes in the reduced ngram to the same function to compute smoothed_prob recursively.

Now that we have can estimate smoothed_prob of any ngram, we then can read the input text and use this smoothed_prob to compute the score p(text|language).



```
# Am Textanfang Leerzeichen als Kontext für die ersten Buchstaben hinzufügen
text_length = len(text)
text = ' '*(L-1) + text

# Berechnung der logarithmierten Wahrscheinlichkeit des Textes für jede Sprache
score = {}
for lang in ngram_prob:
    logp = 0.0
    for i in range(text_length):
        ngram = text[i:i+L]
        logp += math.log2(smoothed_prob(ngram, lang))
    # berechne Crossentropie = negative logarithmierte Wahrscheinlichkeit
    # geteilt durch Textlänge
    score[lang] = -logp / text_length
```

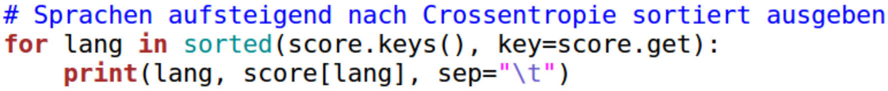
$$p(\text{Er ölt}) = \\ p(\text{E}|\langle s\rangle,\langle s\rangle)\ p(\text{r}|\langle s\rangle,\text{E})\ p(\_\ |\text{E},\text{r})\ p(\text{ö}|\text{r},\_)\ p(\text{l}|\_,\text{ö})\ p(\text{t}|\text{ö},\text{l})\ p(\langle s\rangle|\text{l},\text{t})$$

use the prob of the text to compute a crossentropy score

$$p_L(T) = \prod_{i=1}^{n+1} p_L(a_i|a_{i-k}...a_{i-1})$$

Instead of multiplying the probs, we apply log to each probs and summing them up.

$$lp_L(a_1, ..., a_n) = \sum_{i=1}^{n} log\, p_L(a_i|a_{i-k}, ..., a_{i-1})$$



```
# Sprachen aufsteigend nach Crossentropie sortiert ausgeben
for lang in sorted(score.keys(), key=score.get):
    print(lang, score[lang], sep="\t")
```