

Forum:[Hack Forums](#)

Thread: [Simple custom dictionary generator](#)

Sentyl - April 14, 2025, 6:20 pm

Created a dictionary to generate words using dictionary and probability.

Need a word.data.json file

Here is the code :

Code import json, os, time, random

class Parser:

```
def __init__(self, lang="words_alpha", data={}, dfile="words_alpha.txt"):
    self.LANG, self.DATA, self.DICT = lang, data, dfile
    self.STATS = {"total": 0}
```

```
def statsUpdate(self, ch):
    self.STATS.setdefault(ch, [0, 0])[0] += 1
    self.STATS["total"] += 1
```

```
def getData(self): return self.DATA[self.LANG]
def save(self, d=None): json.dump(d or self.DATA, open("word.data.json", "w", encoding="utf-8"))
def load(self): return json.load(open("word.data.json", "r", encoding="utf-8"))
def loadData(self): self.DATA = self.load()
def reload(self, d=None): self.save(d or self.DATA); self.loadData()
def hasKey(self, k, d): return k not in d
```

class Compiler:

```
def __init__(self, lang="en", data={}, current="start", fixed=False):
    self.LANG, self.DATA, self.CurrentChar, self.Output, self.Fixed = lang, data, current, "", fixed
```

```
def load(self): return json.load(open("word.data.json", "r", encoding="utf-8"))
def loadData(self): self.DATA = self.load()
def values(self): return list(self.DATA[self.LANG]["weights"][self.CurrentChar].values())
def keys(self): return list(self.DATA[self.LANG]["weights"][self.CurrentChar].keys())
```

def parse_main():

```
p = Parser()
p.DICT = input("Enter file (default: dico.en.txt): ") or "dico.en.txt"
p.LANG = input("Enter language (default: same as file): ") or p.DICT
rm = int(input("Delete chars/counts after? 0-Yes 1-No: ") or 0)
if p.hasKey(p.LANG, p.DATA): p.loadData(); p.DATA[p.LANG] = {}; p.save()
```

```
print("Counting...")
p.reload(p.getData().update({"counts": {"start": 0}, "chars": {"start": {}}}))
l = 0
content = open(p.DICT, "r", encoding="utf-8").read().rstrip("\uffff")
for line in content.splitlines():
    i = 0; l += 1; p.statsUpdate(line[0])
    cdata = p.getData()
    cdata["chars"]["start"].setdefault(line[0], 0)
```

```

cdata["chars"]["start"][line[0]] += 1
cdata["counts"]["start"] += 1
for ch in line:
    p.statsUpdate(ch)
    cdata["chars"].setdefault(ch, {})
    cdata["counts"].setdefault(ch, 0)
    if i >= len(line)-1:
        cdata["counts"][ch] += 1
        cdata["chars"][ch].setdefault("end", 0)
        cdata["chars"][ch]["end"] += 1
        break
    next_ch = line[i+1]
    cdata["chars"][ch].setdefault(next_ch, 0)
    cdata["counts"][ch] += 1
    cdata["chars"][ch][next_ch] += 1
    i += 1

```

```

p.reload(); print("Calculating weights...")
cdata = p.getData(); cdata["weights"] = {}
for k in cdata["chars"]:
    cdata["weights"].setdefault(k, {})
    for k2 in cdata["chars"][k]:
        cdata["weights"][k][k2] = cdata["chars"][k][k2] / cdata["counts"][k]
if not rm: del cdata["chars"], cdata["counts"]; print("Removed intermediate data.")
for k in p.STATS:
    if k != "total": p.STATS[k][1] = (p.STATS[k][0]/p.STATS["total"])*100
print("Stats:", p.STATS)
p.save()

```

```

def generate_main():
    out_file = input("Output file: ")
    write = bool(out_file)
    lang = input("Language: ") or "en"
    length_choice = int(input("0: Undetermined | 1: Fixed length: ") or 0)
    compiler = Compiler(lang=lang, fixed=bool(length_choice))
    compiler.loadData()
    length = int(input("Length (default 10): ") or 10) if compiler.Fixed else 0
    count = int(input("Number of words (default 100): ") or 100)
    if write: f = open(out_file, "w")

    for _ in range(count):
        compiler.Output = ""; compiler.CurrentChar = "start"
        while length > len(compiler.Output) or not compiler.Fixed:
            if compiler.Fixed and "end" in compiler.DATA[lang]["weights"][compiler.CurrentChar] and
compiler.DATA[lang]["weights"][compiler.CurrentChar]["end"] == 1.0:
                break
            compiler.DATA[lang]["weights"][compiler.CurrentChar].pop("end", None)
            c = random.choices(compiler.keys(), weights=compiler.values())[0]
            if c == "end": break
            compiler.CurrentChar = c; compiler.Output += c
        print("|", compiler.Output)
        if write: f.write(compiler.Output + "\n")

```

```
if __name__ == "__main__":  
    print("1: Parse | 2: Generate")  
    ch = input(">>> ")  
    parse_main() if ch == "1" else generate_main()
```

Don't now if it useful, but could be.