

## CryptaCount Python Code and Output

### \_\_main\_\_.py

```
from argparse import ArgumentParser, Namespace
from typing import Dict, List

import password_generator

def __create_parser() -> ArgumentParser:
    parser: ArgumentParser = ArgumentParser(
        description="CryptaCount: Password Entropy Analyzer"
    )

    parser.add_argument("--length", type=int, help="Password length", required=True)
    parser.add_argument(
        "--include",
        nargs="+",
        help="Character classes [lower, upper, digits, symbols]",
        required=True,
    )
    parser.add_argument(
        "--exact",
        nargs="*",
        default=[],
        help="Exact counts (e.g., digits=1, symbols=2)",
    )

    return parser

def __create_password_args(args: Namespace) -> password_generator.PasswordArguments:
    length: int = args.length
    included: List[str] = args.include
    exact: Dict[str, int] = __parse_exact_constrains(args.exact)

    return password_generator.PasswordArguments(length, included, exact)
```

```

def __parse_exact_constrains(exact_args: List[str]) -> Dict[str, int]:
    exact: Dict[str, int] = {}

    for item in exact_args:
        if "=" not in item:
            raise ValueError(f"Invalid format for --exact: {item}. Expected key=value")

        key, value = item.split("=")

        if key not in password_generator.CHAR_CLASSES:
            raise ValueError(f"Unknown character class: {key}")

        exact[key] = int(value)

    return exact

def __main() -> None:
    parser: ArgumentParser = __create_parser()

    args: Namespace = parser.parse_args()

    pw_args: password_generator.PasswordArguments = __create_password_args(args)

    search_space: int = password_generator.calculate_search_space(pw_args)
    entropy_bits: float = password_generator.entropy_bits(search_space)
    generated_pw: str = password_generator.generate_password(pw_args)

    password_generator.print_report(pw_args, search_space, entropy_bits, generated_pw)

if __name__ == "__main__":
    __main()

```

## password\_generator.py

```

import math
import secrets

```

```
from dataclasses import dataclass
from typing import Dict, List
```

```
# Character classes available
```

```
CHAR_CLASSES = {
    "lower": "abcdefghijklmnopqrstuvwxyz",
    "upper": "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
    "digits": "0123456789",
    "symbols": "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~",
}
```

```
@dataclass
```

```
class PasswordArguments:
```

```
    length: int
```

```
    included_classes: List[str]
```

```
    exact_counts: Dict[str, int]
```

```
    def __str__(self) -> str:
```

```
        return f"Length: {self.length}
```

```
Included Classes: {self.included_classes}
```

```
Exact Count: {self.exact_counts}"
```

```
def multinomial_coefficient(counts: List[int]) -> int:
```

```
    """Compute multinomial coefficient:  $L! / (k_1! * k_2! * \dots * k_m!)$ ."""
```

```
    total = sum(counts)
```

```
    numer = math.factorial(total)
```

```
    denom = 1
```

```
    for c in counts:
```

```
        denom *= math.factorial(c)
```

```
    return numer // denom
```

```
def check_if_exact_exceed(length: int, req_sum: int) -> None:
```

```
    if req_sum > length:
```

```
        raise ValueError("Sum of exact counts exceeds password length.")
```

```
def calculate_search_space(ps_args: PasswordArguments) -> int:
```

```
    """
```

Calculate the total number of possible passwords (search space N).

- length: password length L
- included\_classes: list of allowed character classes
- exact\_counts: mapping like {"digits": 2}

```
"""
```

```
L: int = ps_args.length
req_sum: int = sum(ps_args.exact_counts.values())
check_if_exact_exceed(L, req_sum)
```

```
sizes: Dict[str, int] = {k: len(CHAR_CLASSES[k]) for k in ps_args.included_classes}
r: int = L - req_sum # remaining positions
```

```
# classes allowed to fill the remaining positions
other_classes: List[str] = [
    c for c in ps_args.included_classes if c not in ps_args.exact_counts
]
```

```
C_other: int = sum(sizes[c] for c in other_classes) if other_classes else 0
```

```
if r > 0 and C_other == 0:
    return 0 # impossible
```

```
# multinomial placement ways
counts_vector: List[int] = list(ps_args.exact_counts.values()) + [r]
place_ways: int = multinomial_coefficient(counts_vector)
```

```
# fill choices for required positions
required_fill: int = 1
for cls, k in ps_args.exact_counts.items():
    required_fill *= sizes[cls] ** k
```

```
# fill choices for remaining positions
other_fill: int = (C_other**r) if r > 0 else 1
```

```
return place_ways * required_fill * other_fill
```

```
def generate_password(ps_args: PasswordArguments) -> str:
    """Generate a password that satisfies exact constraints."""
    L: int = ps_args.length
    req_sum: int = sum(ps_args.exact_counts.values())
```

```

check_if_exact_exceed(L, req_sum)

other_classes: List[str] = [
    c for c in ps_args.included_classes if c not in ps_args.exact_counts
]

other_pool: str = "".join(Char_CLASSES[c] for c in other_classes)

chars: List[str] = []

# Add required characters
for cls, k in ps_args.exact_counts.items():
    for _ in range(k):
        chars.append(secrets.choice(Char_CLASSES[cls]))

# Fill the remaining
r: int = L - req_sum

for _ in range(r):
    chars.append(secrets.choice(other_pool))

secrets.SystemRandom().shuffle(chars)
return "".join(chars)

def entropy_bits(N: int) -> float:
    """Compute entropy (bits)."""
    return math.log2(N) if N > 0 else 0.0

def print_report(
    ps_args: PasswordArguments,
    search_space: int,
    entropy_bits: float,
    generated_pw: str,
) -> None:
    print(ps_args)
    print(f'Search space (N): {search_space:,}')
    print(f'Entropy: {entropy_bits:.2f} bits')
    print(f'Sample password: {generated_pw}')

```

```

def __example() -> None:
    print("Example Run:")

    ps_args: PasswordArguments = PasswordArguments(
        8, ["lower", "upper", "digits"], {"digits": 2}
    )

    N = calculate_search_space(ps_args)
    pw = generate_password(ps_args)
    H = entropy_bits(N)

    print_report(ps_args, N, H, pw)

if __name__ == "__main__":
    __example()

```

```

kharl@kharl-VivoBook-ASUSLaptop-X515EA-X515EA:~/Desktop/Programming/Python/projects/cryptacount$ python3 cryptacount --length 12 --include lower upper digits
Length: 12
Included Classes: ['lower', 'upper', 'digits']
Exact Count: {}
Search space (N): 3,226,266,762,397,899,821,056
Entropy: 71.45 bits
Sample password: pUXAKAKRAu6W
kharl@kharl-VivoBook-ASUSLaptop-X515EA-X515EA:~/Desktop/Programming/Python/projects/cryptacount$ █

-----
Length: 7
Included Classes: ['digits', 'lower']
Exact Count: {'lower': 3}
Search space (N): 6,151,600,000
Entropy: 32.52 bits
Sample password: 3f0n5m4
kharl@kharl-VivoBook-ASUSLaptop-X515EA-X515EA:~/Desktop/Programming/Python/projects/cryptacount$ █

```

```

kharl@kharl-VivoBook-ASUSLaptop-X515EA-X515EA:~/Desktop/Programming/Python/projects/cryptacount$ python3 cryptacount --length 12 --include lower upper digits
Length: 12
Included Classes: ['lower', 'upper', 'digits']
Exact Count: {}
Search space (N): 3,226,266,762,397,899,821,056
Entropy: 71.45 bits
Sample password: pUXAKAKRAu6W
kharl@kharl-VivoBook-ASUSLaptop-X515EA-X515EA:~/Desktop/Programming/Python/projects/cryptacount$ █

```