3 Funkcije

Prvi organizacioni alat koji programeri koriste u Pythonu jeste *funkcija*. Kao i u drugim programskim jezicima, funkcije omogućavaju da izdelite velike programe na manje, jednostavnije delove, sa imenima koja predstavljaju njihovu namenu. One poboljšavaju preglednost i daju pristupačniji kod. Moguće ih je ponovo koristiti i prerađivati.

Funkcije u Pythonu imaju razne dodatne mogućnosti koje programerima olakšavaju život. Neke su slične mogućnostima u drugim programskim jezicima, ali mnoge su jedinstvene za Python. Ti dodaci mogu učiniti namenu funkcije očiglednijom. Oni uklanjaju šum i pojašnjavaju nameru pozivaoca. Mogu znatno smanjiti broj sitnih grešaka koje se teško pronalaze.

Tema 19: Nikada ne raspakujte više od tri promenljive kada funkcije vraćaju više vrednosti

Jedna posledica sintakse za raspakivanje (Tema 6: Dajte prednost raspakivanju sa višestrukom dodelom u odnosu na indeksiranje) jeste to što omogućava Pythonovim funkcijama da naizgled vraćaju više vrednosti. Recimo da pokušavam da utvrdim razne statističke podatke za populaciju aligatora. Data mi je lista dužina (`lengths`) i treba da izračunam najmanju i najveću dužinu u populaciji. Ovde to radim jednom funkcijom koja naizgled vraća dve vrednosti:

```
def get_stats(numbers):
    minimum = min(numbers)
    maximum = max(numbers)
    return minimum, maximum

lengths = [63, 73, 72, 60, 67, 66, 71, 61, 72, 70]

minimum, maximum = get_stats(lengths)  # Dve povratne vrednosti

print(f'Min: {minimum}, Max: {maximum}')
```

```
>>>
Min: 60, Max: 73
```

Ovo radi tako što se više vrednosti vraća zajedno u n-torki sa dve stavke. Kod za pozivanje potom raspakuje vraćenu n-torku dodeljujući vrednosti dvema promenljivama. Ovde koristim još jednostavniji primer da pokažem kako naredba za raspakivanje i funkcija s više rezultata rade na isti način:

```
first, second = 1, 2
assert first == 1
assert second == 2

def my_function():
    return 1, 2

first, second = my_function()
assert first == 1
assert second == 2
```

Više vraćenih vrednosti može da primi izraz sa zvezdicom za raspakivanje sa hvatanjem svih vrednosti (Tema 13: Dajte prednost raspakivanju sa hvatanjem u odnosu na isecanje). Na primer, recimo da mi treba još jedna funkcija koja izračunava koliko je svaki aligator velik u odnosu na prosek za populaciju. Ta funkcija vraća listu odnosa, ali mogu da prihvatim najdužu i najkraću stavku pojedinačno koristeći izraz sa zvezdicom za srednji deo liste:

```
def get_avg_ratio(numbers):
    average = sum(numbers) / len(numbers)
    scaled = [x / average for x in numbers]
    scaled.sort(reverse=True)
    return scaled

longest, *middle, shortest = get_avg_ratio(lengths)

print(f'Longest:  {longest:>4.0%}')
print(f'Shortest: {shortest:>4.0%}')

>>>
Longest:  108%
Shortest:  89%
```

Zamislite sada da se zahtevi programa promene i da treba da utvrdim i prosečnu dužinu, medijanu dužina i veličinu populacije aligatora. To mogu da uradim proširujući funkciju `get_stats` tako da izračunava i te statističke podatke i vrati ih u n-torci rezultata koju raspakuje pozivalac:

```
def get_stats(numbers):
    minimum = min(numbers)
```

```
    maximum = max(numbers)
    count = len(numbers)
    average = sum(numbers) / count

    sorted_numbers = sorted(numbers)
    middle = count // 2
    if count % 2 == 0:
        lower = sorted_numbers[middle - 1]
        upper = sorted_numbers[middle]
        median = (lower + upper) / 2
    else:
        median = sorted_numbers[middle]

    return minimum, maximum, average, median, count

minimum, maximum, average, median, count = get_stats(lengths)

print(f'Min: {minimum}, Max: {maximum}')
print(f'Average: {average}, Median: {median}, Count {count}')

>>>
Min: 60, Max: 73
Average: 67.5, Median: 68.5, Count 10
```

Sa ovim kodom postoje dva problema. Prvi, sve povratne vrednosti su numeričke, pa je previše lako slučajno im izmeniti raspored (npr, zameniti prosečnu dužinu i medijanu), a to može dovesti do programskih grešaka koje će se kasnije teško uočiti. Korišćenje velikog broja povratnih vrednosti izuzetno je podložno greškama:

```
# Ispravno:
minimum, maximum, average, median, count = get_stats(lengths)

# Ups! Medijana i prosek su zamenjeni:
minimum, maximum, median, average, count = get_stats(lengths)
```

Drugi, red koji poziva funkciju i raspakuje vrednosti dugačak je i verovatno će morati da bude prelomljen na neki od više mogućih načina (zbog stila PEP8; Tema 2: Pridržavajte se smernica za stil PEP 8), što umanjuje preglednost:

```
minimum, maximum, average, median, count = get_stats(
    lengths)

minimum, maximum, average, median, count = \
    get_stats(lengths)
```

```
(minimum, maximum, average,
 median, count) = get_stats(lengths)

(minimum, maximum, average, median, count
    ) = get_stats(lengths)
```

Da biste izbegli te probleme, nikada nemojte koristiti više od tri promenljive prilikom raspakivanja višestrukih rezultata funkcije. To bi mogle biti pojedinačne vrednosti iz tročlane n-torke, dve promenljive i jedan izraz sa zvezdicom za hvatanje svih vrednosti, ili bilo šta kraće. Ako treba da raspakujete više vrednosti od toga, bolje bi bilo da definišete jednostavnu klasu ili `namedtuple` (Tema 37: Spajajte klase umesto da ugnežđujete više nivoa ugrađenih tipova) i da zadate da funkcija vraća njenu instancu.

Zapamtite

- Možete zadati da funkcije vraćaju više vrednosti tako što ćete ih postaviti u n-torku i podesiti pozivaoca tako da koristi prednosti Pythonove sintakse za raspakivanje.
- Više vrednosti koje vraća funkcija mogu raspakovati i izrazi sa zvezdicom za hvatanje svih vrednosti.
- Raspakivanje u četiri ili više promenljivih podložno je greškama i treba ga izbegavati; bolje je da povratna vrednost bude mala klasa ili instanca `namedtuple`.

Tema 20: Dajte prednost generisanju izuzetaka u odnosu na vraćanje `None`

Kada pišu pomoćne funkcije na Pythonu, programeri su skloni da daju posebno značanje povratnoj vrednosti `None`. U nekim situacijama, čini se da to ima smisla. Na primer, recimo da pišem pomoćnu funkciju koja jedan broj deli drugim brojem. U slučaju deljenja nulom, vraćanje vrednosti `None` čini se prirodnim jer je rezultat nedefinisan:

```
def careful_divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return None
```

Kod koji koristi ovu funkciju može ispravno da protumači vraćenu vrednost:

```
x, y = 1, 0
result = careful_divide(x, y)
if result is None:
    print('Invalid inputs')
```