Functions

Bu fəsildə siz **functions** yazmağı öyrənəcəksiniz. **Function** – müəyyən bir işi yerinə yetirmək üçün yaradılmış adlandırılmış kod blokudur. Əgər proqramınızda müəyyən bir işi bir neçə dəfə yerinə yetirmək istəyirsinizsə, həmin kodu təkrar-təkrar yazmaq əvəzinə, sadəcə həmin **function**-u çağırırsınız. **Function** çağırışı Python-a həmin funksiyanın içindəki kodu icra etməsini deyir. **Functions** istifadə etmək proqramları daha asan yazmağa, oxumağa, test etməyə və düzəltməyə imkan verir.

Function-a Məlumat Ötürmək

- ★ Siz **function**-lara məlumat ötürmək yollarını da öyrənəcəksiniz.

- ★ Bəzi **functions** yalnız məlumat göstərmək üçün yazılır, bəziləri isə məlumatı işləyib dəyər (və ya dəyərlər) qaytarır.

- ★ Həmçinin **functions**-ları ayrıca fayllarda (**modules**) saxlamaq olar, bu isə əsas proqram faylını təşkil etməyə kömək edir.

Function Tərifi

Sadə bir nümunə: **greet_user()** funksiyası istifadəçiyə salam verir.

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

`def greet_user():` sətiri Python-a deyir ki, siz **function** tərif edirsiniz.

Parentheses (()) içində bu funksiya üçün lazım olan məlumatlar yerləşir. Bu nümunədə heç bir məlumat lazım olmadığından parentheses boşdur.

`"""Display a simple greeting."""` sətiri **docstring** adlanır və funksiyanın nə iş gördüyünü izah edir.

`print("Hello!")` sətiri isə funksiyanın icra etdiyi tək əməliyyatdır.

Funksiyanı istifadə etmək üçün sadəcə onu çağırmaq kifayətdir: `greet_user()`.

Function-a Məlumat Ötürmək
Funksiya azacıq dəyişdirilərək istifadəçini adı ilə salamlayacaq şəkildə yazıla bilər.

```
def greet_user(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user('jesse')
```

`username` funksiya daxilində **parameter**-dir – funksiyanın işi üçün lazım olan məlumat növü.

`'jesse'` isə **argument**-dir – funksiyaya ötürülən məlumat.

`greet_user('jesse')` çağırışı funksiyaya `'jesse'` məlumatını ötürür.

Arguments və Parameters
Function tərifindəki **parameter** – funksiyanın işini görməsi üçün lazım olan dəyişəndir.

Function çağırışında ötürülən dəyər isə **argument** adlanır.

Misal:

```
def greet_user(username):   # username → parameter
    print(f"Hello, {username.title()}!")

greet_user('jesse')         # 'jesse' → argument
```

Bəzən insanlar **arguments** və **parameters** sözlərini qarışdırır. Sürpriz olmasın, çünki **parameter** funksiyanın tərifində, **argument** isə çağırışda olur.

Multiple Arguments

Bir **function** tərifində birdən çox **parameter** ola bilər, buna görə **function call** zamanı birdən çox **argument** ötürmək lazım ola bilər.

Arguments-ı ötürməyin bir neçə yolu var:

1. **Positional arguments** – **arguments** function-dakı **parameters**-in yazıldığı sıraya görə uyğunlaşır.

2. **Keyword arguments** – hər **argument** bir **variable name** və dəyər cütü kimi verilir.

3. **Lists və dictionaries** – dəyərlər **function**-a ötürülə bilər.

Positional Arguments (Pozitsional Argumentlər)

Function çağıranda Python hər **argument**-i **function**-dakı uyğun **parameter**-lə uyğunlaşdırır.

Ən sadə üsul – **arguments**-i **parameters**-in sıraya görə verməkdir

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is
{pet_name.title()}.")

describe_pet('hamster', 'harry')
```

`animal_type` → bir növ heyvan, `pet_name` → heyvanın adı.

Function call: `'hamster'` → `animal_type`, `'harry'` → `pet_name`.

Multiple Function Calls (Birdən çox Function Çağırışı)

Eyni function-u istənilən sayda çağırmaq olar:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is
{pet_name.title()}.")

describe_pet('hamster', 'harry')
```

```
describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
```

Order Matters (Sıranın Önəmi)
Positional arguments istifadə edərkən **arguments**-in sırasi düzgün olmalıdır.

Səhv sıra gözlənilməz nəticə verə bilər.

Keyword Arguments (Açar-Sözlü Argumentlər)
Keyword argument – bir **name=value** cütü.

Bu üsul argumentlərin doğru sırada olub-olmaması ilə maraqlanmadan istifadə edilə bilər.

```
describe_pet(animal_type='hamster', pet_name='harry')
```

Keyword arguments sırası əhəmiyyətli deyil:

```
describe_pet(pet_name='harry', animal_type='hamster')
```

Default Values (Default Dəyərlər)
Function yazarkən **parameter** üçün default value təyin etmək olar.

Əgər function call-da argument verilirsə, Python onu istifadə edir; əks halda default value istifadə olunur.

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is
{pet_name.title()}.")

describe_pet(pet_name='willie')
```

Equivalent Function Calls (Eyni Nəticə verən Function Çağırışları)
Positional arguments, keyword arguments və default values birlikdə istifadə oluna bilər.

Misal:

```
# Dog named Willie
```

```
describe_pet('willie')
describe_pet(pet_name='willie')

# Hamster named Harry
describe_pet('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
```

Hamısı eyni çıxışı verir.

Avoiding Argument Errors (Argument Xətalarından Qaçınmaq)

Function istifadə edərkən unmatched arguments xətası ilə qarşılaşa bilərsiniz.

Məsələn, heç bir argument verməmək:

describe_pet()
 # TypeError: describe_pet() missing 2 required positional arguments: 'animal_type' and 'pet_name'

Return Values (Return Dəyərləri)

1️⃣ Return Value nədir?

- Function həmişə nəticəsini birbaşa ekrana göstərmək məcburiyyətində deyil.
- Əvəzinə, məlumatı işləyib bir dəyər (və ya dəyərlər) qaytara bilər.
- Return value – funksiyanın qaytardığı dəyərdir.
- `return` statement function-un içindəki dəyəri çağırıldığı yerə geri göndərir.
- Return values ilə proqramın əsas işlərini funksiyalara verib, əsas kodu daha səliqəli yaza bilərsiniz.

Sadə Return Value

Misal: birinci və son adı alıb tam adı qaytaran function:

```
def get_formatted_name(first_name, last_name):
```

```
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
```

`full_name` dəyişəni `first_name` və `last_name`-i birləşdirir.

`return` statement funksiyanı çağıran sətirə (`musician`) dəyəri qaytarır.

3️⃣ Optional Arguments (İstəyə bağlı argumentlər)

Bəzən argumenti istəyə bağlı etmək faydalıdır.

Bunun üçün **default value** istifadə olunur.

```
def get_formatted_name(first_name, last_name,
middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = f"{first_name} {middle_name}
{last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

`middle_name` default olaraq boş string (`''`) verilir, yəni verilən deyilsə, ignored olunur.

4️⃣ Function Return Dictionary

Function hər hansı dəyəri qaytara bilər, o cümlədən **lists** və **dictionaries**.

```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a
person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

None – dəyişən üçün xüsusi placeholder dəyərdir.

Bu funksiya adı həmişə saxlayır, amma əlavə məlumatları optional olaraq əlavə etmək mümkündür.

5️⃣ Function ilə while Loop istifadə etmək
Functions-ları bütün Python strukturları ilə istifadə etmək olar.

Misal: istifadəçini **while loop** ilə salamlayan proqram:

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")
    f_name = input("First name: ")
    if f_name == 'q':
        break
    l_name = input("Last name: ")
    if l_name == 'q':
```

```
        break
    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

`while True:` sonsuz loop yaradır.

Hər iki input üçün q daxil edilərsə **break** statement ilə çıxılır.

Proqram istifadəçi q daxil edənə qədər işləyir.

Output:
Please tell me your name:
(enter 'q' at any time to quit)
First name: eric
Last name: matthes
Hello, Eric Matthes!
Please tell me your name:
(enter 'q' at any time to quit)
First name: q

Passing a List (List-i Funksiyaya ötürmək)

1️⃣ List-i Funksiyaya ötürmək

- Tez-tez list-i function-a ötürmək faydalıdır: adlar, ədədlər və ya daha mürəkkəb obyektlər (məsələn, dictionary) ola bilər.

- List function-a ötürüldükdə, funksiya onun içindəkilərə birbaşa çıxış əldə edir.

Misal: istifadəçilərə salam göndərmək:

```
def greet_users(names):
```

```
    """Print a simple greeting to each user in the
list."""
    for name in names:
        msg = f"Hello, {name.title()}!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

`greet_users()` function list-i alır və hər istifadəçiyə salam verir.

Bu function-u istənilən vaxt çağırıb fərqli list istifadə edə bilərsiniz.

2️⃣ Function daxilində List-i dəyişdirmək

- List-i function-a ötürdükdə, function onu dəyişdirə bilər.
- Dəyişikliklər daimi olur.

Misal: 3D modelləri çap edən şirkət:

```
# Çap olunacaq modellər

unprinted_designs = ['phone case', 'robot pendant',
'dodecahedron']

completed_models = []



while unprinted_designs:

    current_design = unprinted_designs.pop()

    print(f"Printing model: {current_design}")

    completed_models.append(current_design)
```

```
print("\nThe following models have been printed:")
for completed_model in completed_models:
    print(completed_model)
```

`unprinted_designs` list-i boşalana qədər loop işləyir, hər model `completed_models`-ə əlavə olunur.

Output:
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case

Funksiyalara bölmək (Function Refactoring)

- Eyni kodu iki function-a bölmək:

```
def print_models(unprinted_designs, completed_models):
    """Simulate printing each design until none are
left."""
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
```

```
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)

unprinted_designs = ['phone case', 'robot pendant',
'dodecahedron']
completed_models = []

print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

`print_models()` – çap etmə işini görür.

`show_completed_models()` – tamamlanmış modelləri göstərir.

Kod daha təşkilatlanmış və oxunaqlıdır.

Output:
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case

Function-un List-i dəyişdirməsini önləmək
Bəzən function-un list-i dəyişdirməsini istəməyə bilərsiniz.

Bunun üçün original list əvəzinə copy göndərin:
print_models(unprinted_designs[:], completed_models)

- [:] slice notation ilə list-in kopyasını yaradır.

- Function kopya üzərində işləyir, original list dəyişmir.

Qeyd:

- Adətən, böyük list-lərlə işləyərkən birbaşa original list-i ötürmək daha effektivdir (vaxt və yaddaş baxımından).

- Copy yalnız spesifik səbəblər varsa istifadə olunur.

Passing an Arbitrary Number of Arguments (İstənilən sayda argument ötürmək)

1️⃣ İstənilən sayda positional argument

- Bəzən function-un neçə argument alacağını əvvəlcədən bilmirik.

- Python buna imkan verir: function istənilən sayda argument toplaya bilər.

Misal: pizza hazırlamaq function-u:

```
def make_pizza(*toppings):
    """Print the list of toppings that have been
requested."""
    print(toppings)


make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra
cheese')
```

`*toppings` – Python-a **tuple** yaratmağı və bütün dəyərləri ora yerləşdirməyi bildirir.

Output:
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')

İndi `print()`-i loop ilə əvəzləyə bilərik:

```
def make_pizza(*toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

Output:
Making a pizza with the following toppings:
- pepperoni
Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese

Positional və arbitrary arguments qarışdırmaq
Əgər function həm **normal**, həm də arbitrary argument qəbul edirsə, arbitrary argument **sonda** olmalıdır.

Python əvvəlcə positional və keyword argument-ları uyğunlaşdırır, qalanları tuple-a yığır.

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
```

```
    for topping in toppings:
        print(f"- {topping}")

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra
cheese')
```

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese

`size` – pizza ölçüsünü alır.

`*toppings` – bütün əlavə ingredientləri tuple kimi toplayır.

İstənilən sayda keyword arguments

- Bəzən function-un hansı məlumat alacağını əvvəlcədən bilmirik.
- Bu halda **arbitrary keyword arguments** istifadə olunur (`**kwargs`).

Misal: istifadəçi profilini yaratmaq:

```
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we
know about a user."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
```

```
                                location='princeton',
                                field='physics')
print(user_profile)
```

{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'last_name': 'einstein'}

`**user_info` – Python-a **dictionary** yaratmağı və bütün əlavə key-value cütlərini ora yerləşdirməyi bildirir.

Summary
`*args` → arbitrary positional arguments

`**kwargs` → arbitrary keyword arguments

`*args` həmişə sonradan əlavə positional argumentları toplamaq üçün istifadə olunur.

`**kwargs` istənilən sayda key-value cütünü dictionary şəklində toplamaq üçün istifadə olunur.

Storing Your Functions in Modules (Funksiyaları modullarda saxlamaq)

1️⃣ Funksiyaları modulda saxlamaq

- Funksiyaları ayrıca faylda saxlamaq kodunuzu daha oxunaqlı edir.
- Bu fayla **module** deyilir (`.py` uzantılı fayl).
- Sonra `import` statement ilə həmin modulun funksiyalarını əsas proqramınıza gətirə bilərsiniz.

Məsələn, pizza funksiyasını modulda saxlayaq (pizza.py adinda yeni file yaradin):

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")
```

İndi başqa fayl yaratmaqla bu funksiyanı istifadə edə bilərik:

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers',
'extra cheese')
```

`import pizza` → Python `pizza.py` faylını oxuyur və funksiyaları proqramınıza əlavə edir.

Funksiyanı çağırmaq üçün `pizza.make_pizza()` istifadə olunur.

Müəyyən funksiyanı import etmək

Bütün modulu deyil, sadəcə bir funksiyanı da gətirə bilərsiniz:

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")
import pizza
```

```
pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers',
'extra cheese')
from pizza import make_pizza

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra
cheese')
```

İndi **dot notation** lazım deyil, birbaşa `make_pizza()` çağırırıq.

Funksiyaya alias vermək

Əgər funksiyanın adı uzun və ya başqa bir funksiya ilə üst-üstə düşürsə, ona alias verə bilərsiniz:

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers',
'extra cheese')


from pizza import make_pizza as mp

mp(16, 'pepperoni')
mp(12, 'mushrooms', 'green peppers', 'extra cheese')
```

`mp()` funksiyanı əvəz edir.

Modulun adını alias ilə istifadə etmək

Modula da qısa ad verə bilərsiniz:

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")

import pizza as p

p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra
cheese')
```

Bu üsul funksiyaların orijinal adını saxlayır, amma daha qısa yazmaq mümkündür.

Bütün funksiyaları import etmək

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the
following toppings:")
    for topping in toppings:
        print(f"- {topping}")

from pizza import *

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra
cheese')
```

Qeyd: Böyük modullarda bu üsul qarışıqlığa səbəb ola bilər, çünki funksiya adları üst-üstə düşə bilər.

Funksiyaları düzgün stil ilə yazmaq

- Funksiya adları **aşağı hərflər + underscore** ilə yazılmalıdır: `function_name`

- Modul adları da eyni konvensiyaya riayət etməlidir.

- Hər funksiyanın docstring ilə qısa izahı olmalıdır.