

Functions

Defining and Using Functions

$f(x)$

SoftUni Team

Technical Trainers



SoftUni



Software University

<https://softuni.bg>

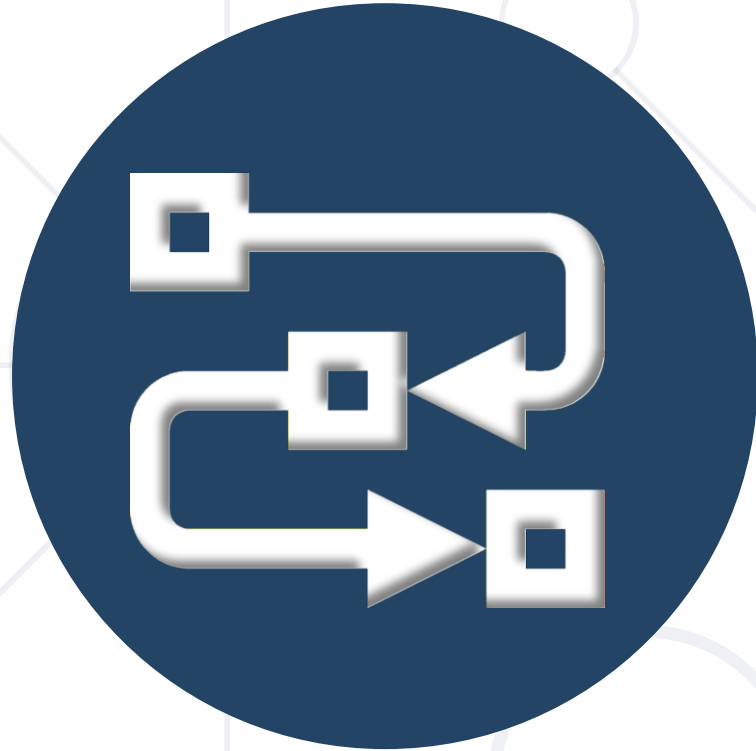
sli.do

#fund-python

Table of Contents

1. **Functions** Overview
2. **Declaring** and **Invoking** Functions
3. **Return Values**
4. **Parameters** vs **Arguments**
5. **Lambda** Functions





Functions Overview

Declaring and Invoking Functions

Functions

- Function == named piece of code
 - Can take parameters and return result

Use snake_case

Function
parameter

```
def function_name(parameter: type):  
    statement(s)
```

Type of the
parameter



Why Use Functions?

- More **manageable programming**
 - Splits large problems into small pieces
 - Better organization of the program
 - Improves code readability
 - Improves code understandability
- Avoiding **repeating code**
 - Improves code maintainability
- Code **reusability**
 - Using existing functions several times



- Python has a set of **built-in functions** that we can call at **any time**
- List of some built-in functions

```
abs()  
min()  
max()  
round()
```

```
sum()  
filter()  
map()  
sorted()
```

Problem: Absolute Values

- Write a program that
 - Receives a sequence of numbers, separated by a **single space**
 - **Prints** their **absolute value** as a list

1 2.5 -3 -4.5



[1.0, 2.5, 3.0, 4.5]

Problem: Absolute Values

```
list_of_strings = input().split()

list_of_numbers = []
for n in list_of_strings:
    number = float(n)
    list_of_numbers.append(number)

list_of_absolute_numbers = []
for n in list_of_numbers:
    absolute_number = abs(n)
    list_of_absolute_numbers.append(absolute_number)

print(list_of_absolute_numbers)
```



Declaring and Invoking Functions


Declaring Function

Function Name

Parameters

```
def print_text(text):  
    print(text)
```

Function
Body

- 
- Using the **def** statement is the most common way to define a function in Python
 - Functions can have **several parameters**
 - It is possible for the function to **not** return a value

Invoking a Function

- Functions are **first** declared, then **invoked** (many times)

```
def print_header():  
    print("This is header")
```

Function
Declaration

- Functions can be **invoked** (called) by their name

```
print_header()
```

Function
Invocation

- A function can be invoked from:

- Other functions

```
def print_header():  
    print_header_top()  
    print_header_bottom()
```

Function invoking
functions

- Itself (recursion)

```
def crash():  
    crash()
```

Function invoking
itself

Function without Parameters

- Executes the code after
- Does not return result

```
def multiply_numbers():  
    result = 5 * 5  
    print(result)  
multiply_numbers() #25
```

Prints result
on the
console



return

Return Values

The Return Keyword

- Functions can return a value that you can use directly:

```
def give_me_five():  
    return 5  
print(give_me_five()) # Print the returned value  
#Out: 5
```

- or save the value for later use:

```
num = give_me_five()  
print(num) #Print the saved returned value  
#Out: 5
```


- If **return** is encountered in the function the function will be exited immediately

```
def give_me_another_five():  
    return 5  
    print('This statement will not be printed.')  
print(give_me_another_five()) #Out: 5
```

- Write a program that **receives a grade** between 2.00 and 6.00 and **prints the corresponding grade in words**
 - Between **2.00** and **2.99** - '**Fail**'
 - Between **3.00** and **3.49** - '**Poor**'
 - Between **3.50** and **4.49** - '**Good**'
 - Between **4.50** and **5.49** - '**Very Good**'
 - Between **5.50** and **6.00** - '**Excellent**'

```
def grades(grade):  
    if grade >= 2.00 and grade <= 2.99 :  
        return 'Fail'  
    elif grade >= 3.00 and grade <= 3.49:  
        return 'Poor'  
    # TODO: Add other conditions
```

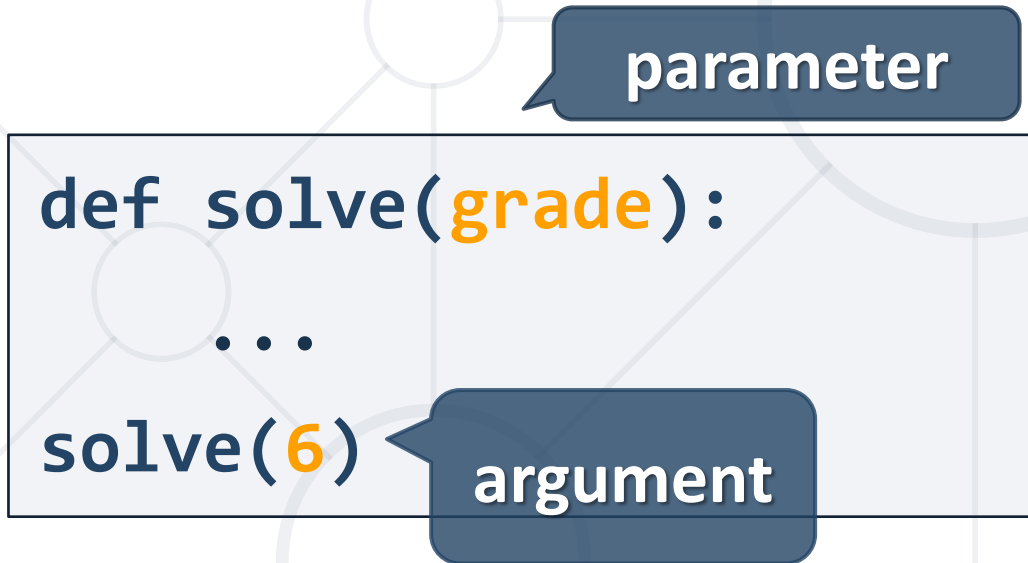


**params
vs
args**

Parameters vs Arguments

Parameters vs Arguments

- **Parameter** is a variable defined in a function definition, while the **argument** is an actual value passed to the function



```
def solve(grade):  
    ...  
solve(6)
```

The diagram illustrates the difference between a parameter and an argument. A light blue box contains the code for a function definition and a function call. A dark blue speech bubble labeled "parameter" points to the word "grade" in the function definition. Another dark blue speech bubble labeled "argument" points to the number "6" in the function call.

- Function arguments can have **default** values
- If the function is called **without the argument**, the argument gets its default value

```
def person(first_name = 'George', last_name = 'Brown'):  
    print(first_name, last_name)  
person('Peter') #'Peter Brown'
```

Keyword (Named) Arguments

- Functions can be called using **keyword arguments**
- When we use keyword/named arguments, it's the **name** that matters, not the **position**

```
def area(width, height):  
    return width * height  
print(area(height = 2, width = 1))
```

Problem: Calculations

- Write a function that **receives three parameters** and calculates a result depending on the operator
- The operator can be '**multiply**', '**divide**', '**add**', '**subtract**'
- The input comes as three parameters - two **integers** and an operator as a **string**

5, 10, 'multiply'



25


```
def solve(a,b,operator):  
    result = None  
    if operator == 'multiply':  
        result = a * b  
    elif operator == 'divide':  
        result = a / b  
    # TODO : other cases  
    return result  
print(solve(5,10,'multiply')) # 50
```



Lambda Functions

Lambda Definition

- Lambda is an **anonymous one-time** function
 - Like a function, it can take a parameter and return a result



key word


arguments

expression

```
x = lambda a: a + 10  
print(x(5))  # 15
```

Lambda Example

- It can take multiple parameters



```
x = lambda a, b: a * b
print(x(3, 4))  # 12
```

```
full_name = lambda first, last: f'I am {first} {last}'
result = full_name('Guido', 'van Rossum')
print(result)  # I am Guido van Rossum
```

Problem: Repeat String

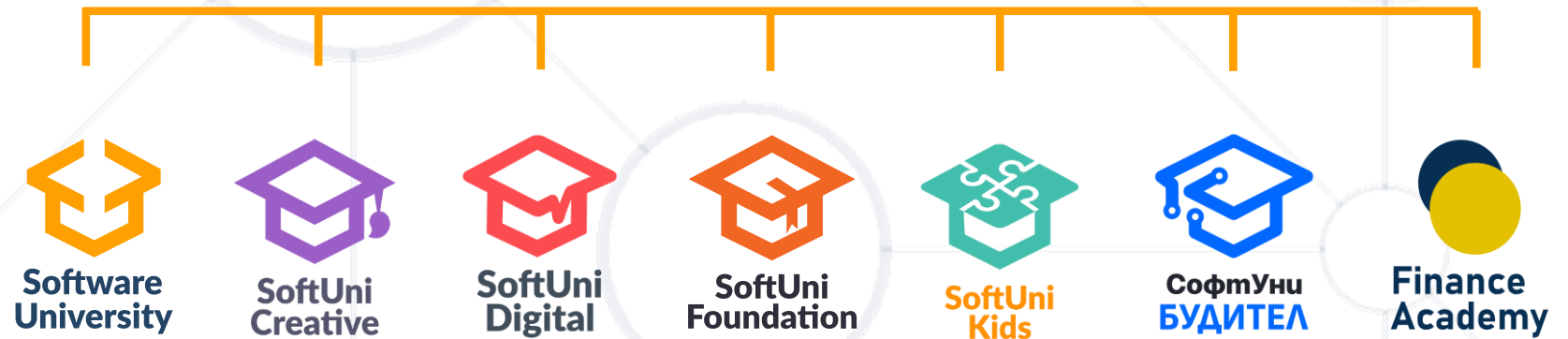
- Write a function which **receives** a **string** and a **counter n**
- The function should return a **new string** – the **result** of repeating the old string **n** times
- Print the result on the console



- Break large programs into simple **functions** that solve small sub-problems
- Consist of **declaration** and **body**
- Are invoked by their **name**
- Can accept **parameters**



Questions?



SoftUni Diamond Partners



THE CROWN IS YOURS



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, softuni.org
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

