Functions in Python

Table of contents

0.1 Big O Notation

Big O notation is used to describe the performance or complexity of an algorithm. It specifically describes the **worst-case** scenario, and can be used to describe the execution time required or the space used by an algorithm.

0.2 Functions

- Functions are **reusable** blocks of code that perform a specific task
- They can take input parameters and return output values
- Functions are essential in modular programming, as they help organize code and make it more readable

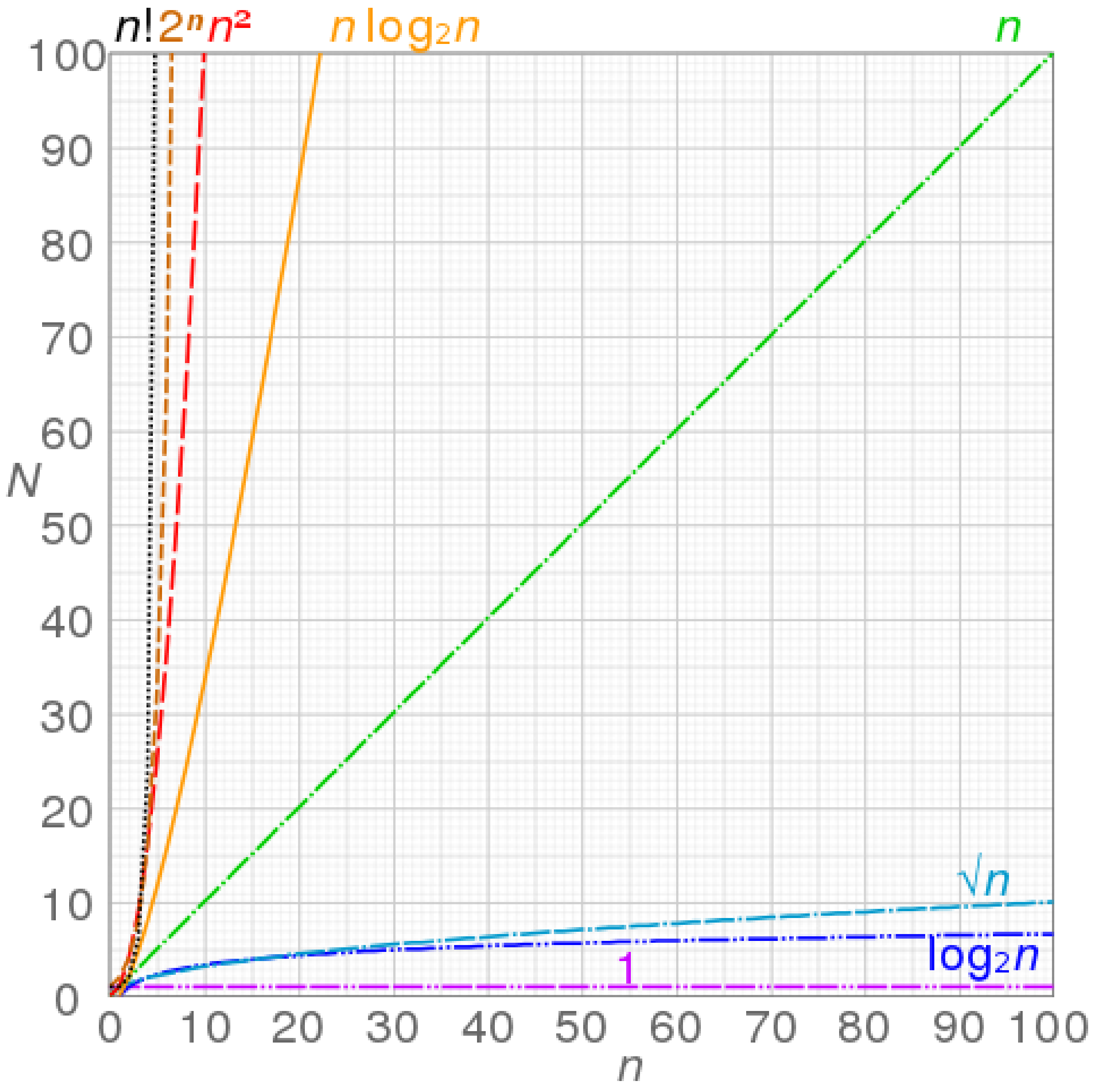

Figure 1: Comparison computational complexity

0.3 Defining a Function

- To define a function in Python, use the keyword `def` followed by the function name and input parameters in parentheses
- The function body is **indented** below the header line
- Use the keyword `return` to specify the output value(s) of the function

```python
def add_numbers(x, y):
    result = x + y
    return result

type(add_numbers)
```

```
function
```

0.4 Calling a Function

To call a function, use its name followed by input values in parentheses The function returns the output value(s), which can be stored in a variable or used directly

```python
sum = add_numbers(2, 3)
print(sum)
```

```
5
```

0.5 Default Parameter Values

- Functions can have default values for input parameters, which are used when no value is provided
- Default values are specified in the function header

```python
def greet(name, greeting = "Hello"):
    print(greeting + ", " + name)

greet("Alice")
```

```
Hello, Alice
```

```python
greet("Bob", "Hi")
```

```
Hi, Bob
```

0.6 Variable-Length Arguments

- Variable-length arguments allow a function to accept any number of input arguments
- They are useful when the number of input arguments is unknown or can vary

```python
def add_numbers(*args):
    result = 0
    for num in args:
        result += num
    return result

add_numbers
```

```
<function __main__.add_numbers(*args)>
```

```python
add_numbers(1, 2)
```

```
3
```

```python
add_numbers(1, 2, 3)
```

```
6
```

0.7 Function Annotation

```python
def add_numbers(*args):

    """
    Computes the sum of n numbers

    Parameters:
    args: A tuple of numbers

    Returns:
    int: The sum
    """

    result = 0
    for num in args:
        result += num
    return result

help(add_numbers)
```

```
Help on function add_numbers in module __main__:

add_numbers(*args)
    Computes the sum of n numbers

    Parameters:
    args: A tuple of numbers

    Returns:
    int: The sum
```

0.8 Lambda Functions

- Lambda functions are **anonymous** functions that can be defined inline and used immediately
- They are useful for *simple* tasks that don't require a named function
- Lambda functions can only have **one** expression

```
double = lambda x: x * 2
print(double(3))
```

```
6
```

0.9 Recursion

- Recursion is a technique where a function calls itself
- It is useful for solving problems that can be broken down into smaller subproblems

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

print(factorial(5))
```

```
120
```

0.10 Global vs Local Variables

- Global variables are defined outside of any function and can be accessed from anywhere in the program
- Local variables are defined inside a function and can only be accessed within that function

```python
global_var = 10

def my_func():
    local_var = 20
    print(global_var)
    print(local_var)

my_func()
```

```
10
20
```

```python
print(global_var)
```

```
10
```

```
# print(local_var)
```

1 Exercises

1.1 Exercise 1

Python function that takes a list of numbers and returns their product

```
multiply_numbers([2, 3, 4])
```

```
24
```

```
def multiply_numbers(numbers):
  product = 1
  for num in numbers:
    product *= num
  return product
```

1.2 Exercise 2

Write a Python function to find the maximum and minimum numbers in a list

```
find_min_max ([2, 3, 4])
```

```
(2, 4)
```

```
def find_min_max(numbers):
  min_num = numbers[0] # Set the min to the first number in the list
  max_num = numbers[0] # Set the max to the first number in the list
  for num in numbers:
    if num > max_num:
      max_num = num
    elif num < min_num:
      min_num = num
  return (min_num, max_num)
```

```python

find_min_max ([2, 3, 4])
```

```
(2, 4)
```

1.3 Exercise 3

Write a Python function that takes a list of strings and returns the longest string.

```python
find_longest_string(["Vicky", "Cristina", "Barcelona"])
```

```
'Barcelona'
```

```python
def find_longest_string(strings):
  # Set the longest string to first string in the list
  longest_string = strings[0]
  for string in strings:
    # Compare the length of the current string to the longest string so
    ↪  far
    if len(string) > len(longest_string):
      longest_string = string
  return longest_string

find_longest_string(["Vicky", "Cristina", "Barcelona"])
```

```
'Barcelona'
```