Functions. Variable Scope.

Досега си говорихме за:

1. Values & Types
2. Expressions
3. Statements
4. Алгоритми и подход към задачите

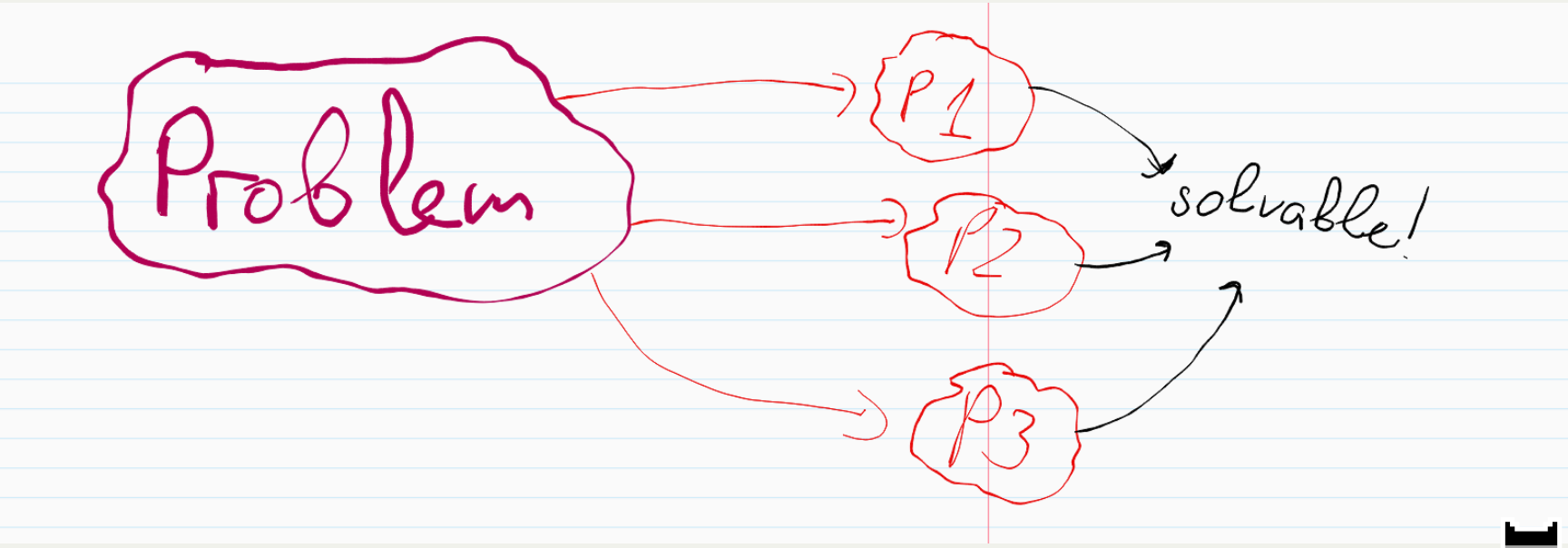
Основните types:

1. Int
2. Float
3. String
4. Boolean
5. NoneType
6. **List**
7. **???**

Програмите, които писахме:

1. Ползват научения синтаксис.
2. Четат потребителски вход.
3. Решават конкретен проблем.
4. print() към конзолата.
5. Често имаме повтарящи се проблеми.

Един проблем се състои от по-малки проблеми.



Примери:

1. Проверяваме дадено Да/Не свойство за цяло число.
2. Правим нещо за всяко число в интервал от цели числа. Например **[1, N].**
3. Броим колко неща в интервал спазват определени свойства?
4. Броим колко неща в списък изпълняват определени условия?

Reflection!

Да се върнем стъпка назад и да помислим какво се случва.

За да решим проблем, който сме решавали преди:

1. Трябва да напишем кода, с който сме го решили преди.
2. Съобразявайки променливите.
3. Защото най-вероятно се намираме в друга програма.
4. Ако един проблем се решава чрез 3 други, това става тежко.

Да разгледаме алгоритъма за решаване на Twin Primes проблема:

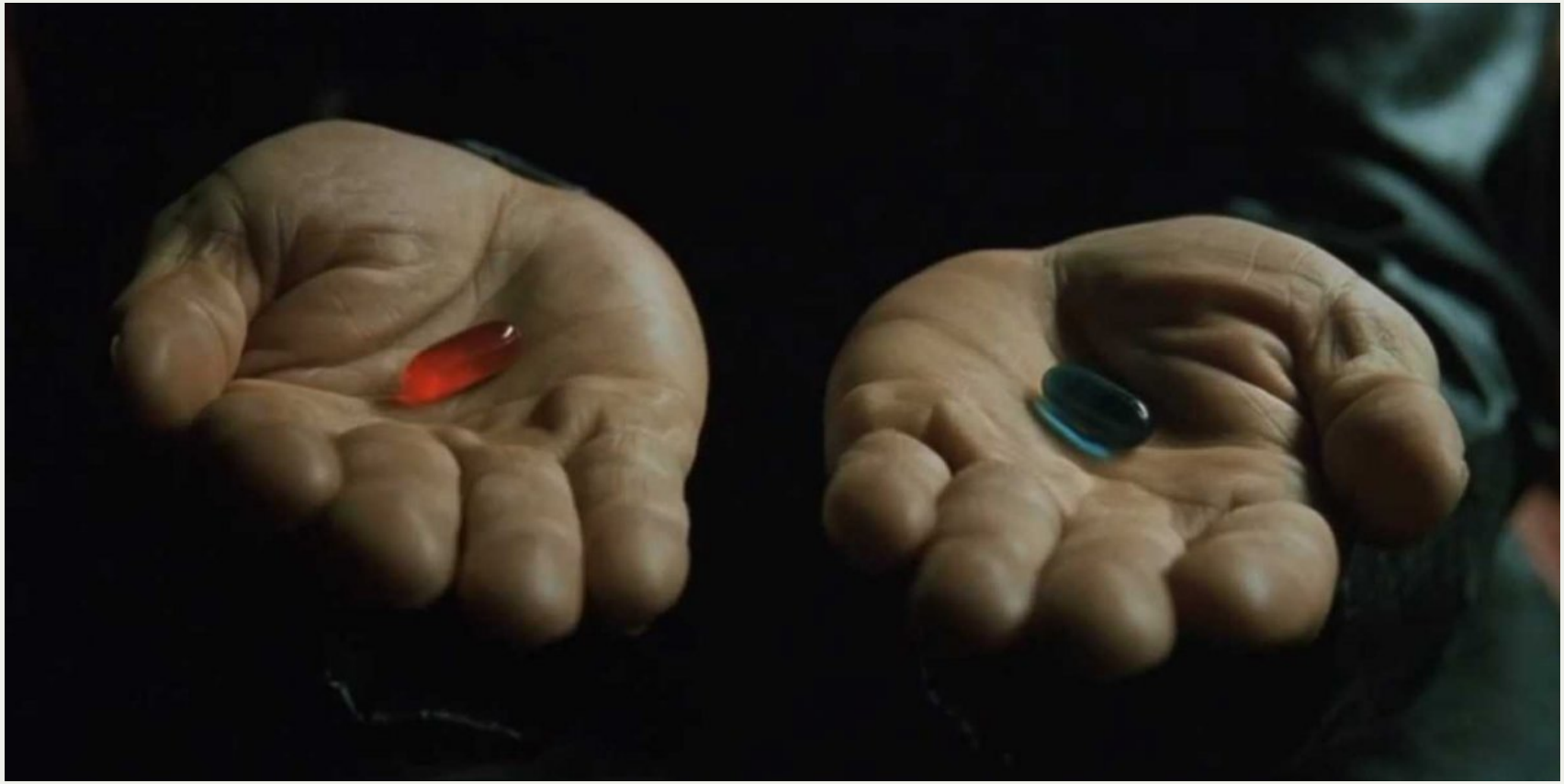
1. Четем p.
2. Създаваме q = p - 2.
3. Създаваме r = p + 2.
4. **Проверяваме дали p, q и r са прости и взимаме решение.**

Тройната проверка изглежда тежка.

1. Евентуално подобрение е да вкараме p, q и r в списък, който да обходим, проверявайки.
2. Но това ще създаде други проблеми.
3. А и ако трябва след това да проверим нещо друго, свързано с прости числа ...

Не трябва да ни е толкова тежко!

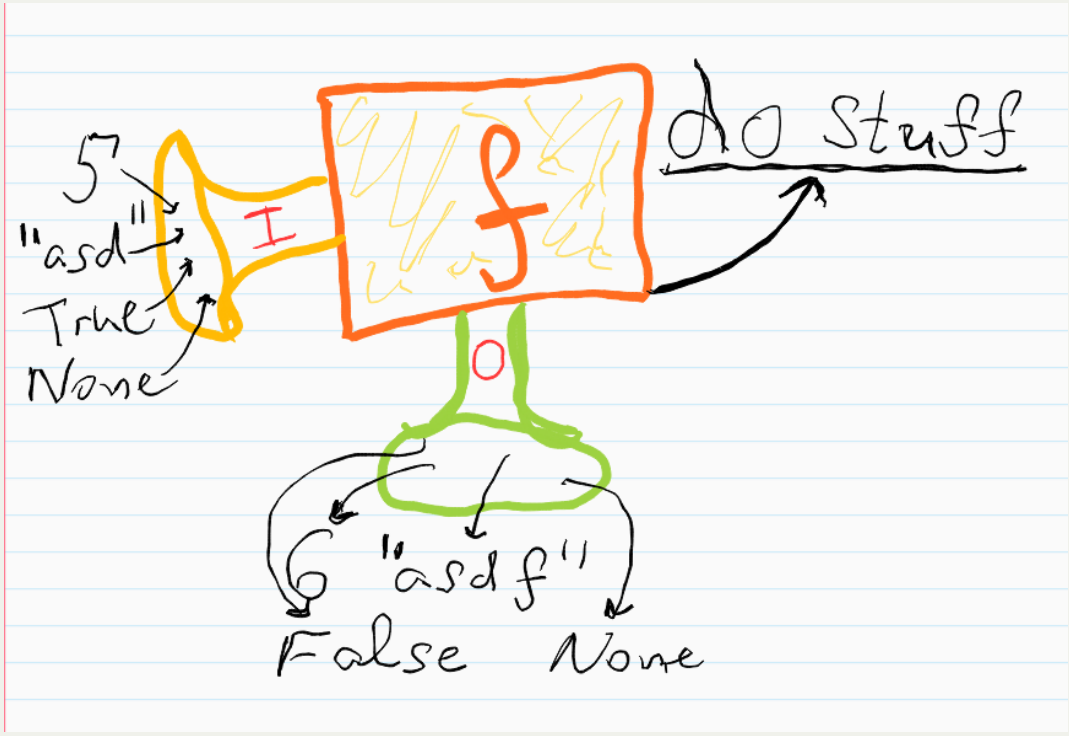
1. Искаме да решим 1 път даден проблем, след което да ползваме това решение навсякъде, където го срещнем.
2. Искаме да пишем по-малко код, без да се повтаряме.
3. Искаме да скрием сложнотията на даден алгоритъм зад хубаво име.




POWER
UNLIMITED POWER!
quickmeme.com

Функции.

Или как да направим програма в програмата.



The real voyage of discovery consists not in seeking new landscapes, but in having new eyes.

~Marcel Proust

Няколко примерни функции:

```
def inc(x):
    return x + 1
```

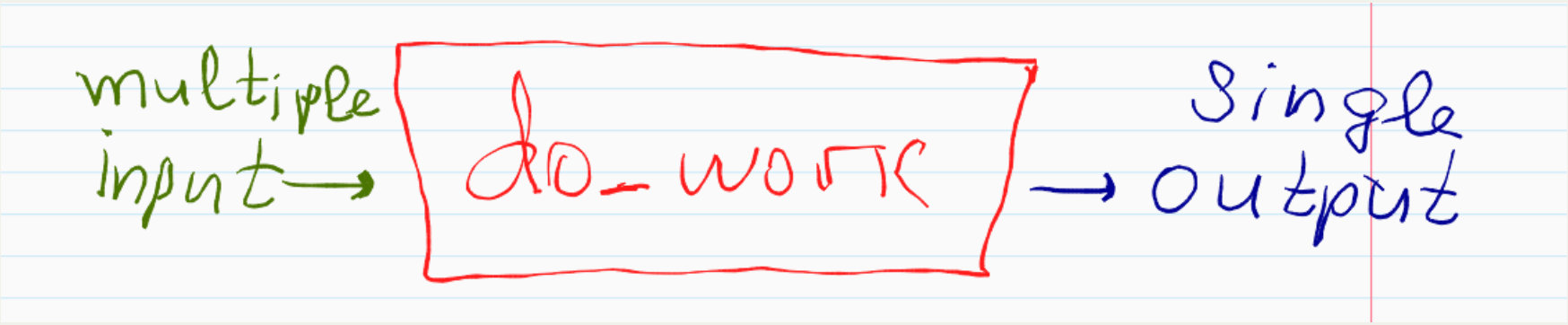
```
def dec(x):
    return x - 1
```

```
def id(x):
    return x
```

Функциите са:

1. Парче код, **скрито зад име**, което решава точно определен проблем.
2. Преизползваеми.
3. **Взимат вход**, решават точно определен проблем и **връщат резултат**.
4. Представляват програма в програмата.
5. **Градивен елемент на нашите програми.**
6. while / if / for са градивен елемент на функциите.

Функциите приемат вход, вършат работа и връщат резултат.



Функциите "връзват" една стойност с друга.

$$f(x) = x^2$$

$$f(1) = 1$$

$$f(2) = 4$$

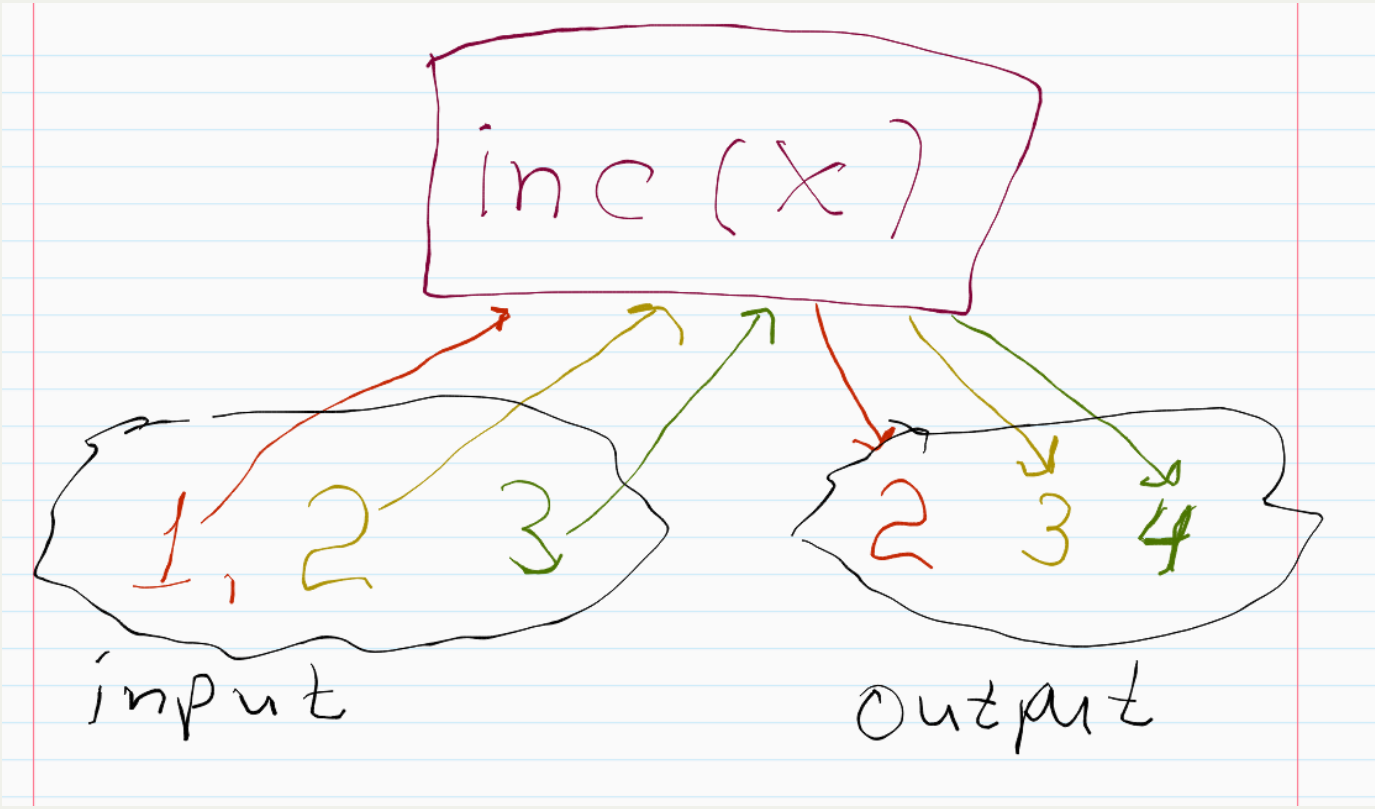
Използване на функции:

```
def inc(x):
    return x + 1
```

```
n = input()
n = int(n)

n_plus_one = inc(n)
print(n_plus_one)
```

Диаграма на inc



Синтаксис на функции

1. `def` е ключова дума за създаване на функция.
2. `inc` е името на функцията. Така ще я извикаме после.
3. `(x)` представлява аргументите на една функция.
4. : винаги го забравяме
5. `return` pace-а пишем **тялото на функцията.**
6. return е ключова дума, с която казваме каква стойност връща нашата функция.

Аргументи на функция

```
def add_3(a, b, c):
    return a + b + c
```

1. Аргументите са **a**, **b** и **c**
2. Те представляват "входа на функцията"
3. Това, което потребителя дава като стойност, за да получи резултат.
4. **a, b и c са променливи за тялото на функцията.**
5. Една функция може да е без, с един, два или много аргументи.

return statement

```
def add_3(a, b, c):
    return a + b + c
```

```
def say_hello(name):
    print("Hello there " + name)
```

1. Не е задължителен за функция.
2. Без него, функцията връща None.
3. С него, функцията връща изразът след него.

Глаголи с функции

1. Дефинираме функция.
2. Извикваме функция.
3. Връщаме стойност от функция.

```
# Call function
a = inc(5)
print(a)

# Just assing function to a variable
b = inc
print(b)
```

Нека да напишем функция, която смята:

$$f(a, b) = a^2 + b^2$$

```
def square_sum(a, b):
    return a ** 2 + b ** 2
```

Функция, която намира най-голямото от 3 числа.

```
def largest_of_3(a, b, c):
    largest = a

    if b >= largest:
        largest = b

    if c >= largest:
        largest = c

    return largest
```

Функция, която намира най-голямото от 3 числа.

1. Няма проблеми да имаме дълго тяло на функцията.
2. Може да използваме всички познати statements вътре.
3. Броят на аргументите е важен. Така потребителят ще знае как да работи с функцията.

Извикването на функции е израз!

1. Извикване на функция с аргумент се свежда до някоя от простите стойности
2. Например: `largest_of_3(3,4,5)`
3. Се оценява до 5.

Може да имаме сложни изрази от функции!

```
largest_of_3(inc(3), inc(4), dec(6))
```

```
square_sum(inc(1), largest_of_3(square_sum(1, 2), 4, 5))
```

Правилата за оценяване на израза са:

1. Python намира най-вътрешното извикване на функция, което може да оцени.
2. Оценява го.
3. Ако има още функции в израза, повтаря стъпка 1)

```
square_sum(inc(1), largest_of_3(square_sum(1, 2), 4, 5))
```

```
square_sum(inc(1), largest_of_3(5, 4, 5))
```

```
square_sum(inc(1), 5)
```

```
square_sum(2, 5)
```

```
29
```

Извикване на функции от други функции.

```
def inc(x):
    return x + 1

def dec(x):
    return x - 1

def do_nothing(x):
    return inc(dec(x))
```

Решете задачите от седмица 3.

Variable Scope

(видимост на променливите)

1. Една променлива може да се вижда на ниво **"цялата програма". Това се нарича глобална променлива.**
2. Една променлива може да се вижда на ниво **"функция". Това се нарича локална променлива.**

Global variables.

```
start = 1
end = 5

product = 1

while start <= end:
    inside_loop = start

    product *= inside_loop
    start += 1

print(product) # 120
print(inside_loop) # 5
```

Global variables.

1. Променливи, дефинирани в програма, се виждат в цялата програма.
2. Дори, когато са дефинирани в if или while statement.
3. Тези променливи се виждат и във функции*.

Local variables

```
def do_something():
    a = 6
    print(a)
    return 42

do_something()
print(a)

# 6
# NameError: name 'a' is not defined
```

Local variables

1. Променлива, дефинира във функция се нарича "локална променлива"
2. Тя се вижда само в дадената функция.
3. Извън функцията - тази променлива не е видима.

Локални и глобални с едно и също име.

```
a = 5

def do_something():
    a = 6
    print(a)

do_something()
print(a)

# 6
# 5
```

Стига толкова!