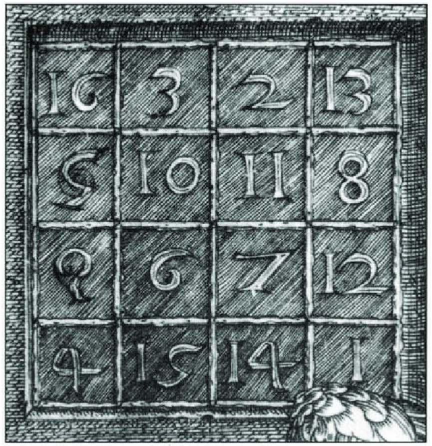
Logic and Discrete Structures -LDS



Course 2

Lecturer Dr. Eng. Cătălin Iapă
e- mail: catalin.iapa@cs.upt.ro
Facebook : Catalin Iapa
cv: Catalin Iapa

What did we do last time?

Demonstrations (Proofs)
Sets
Functions
Properties of Functions
Functions in Programming

What did we do last time?

Sets

- Element of a set - Subsets

Functions

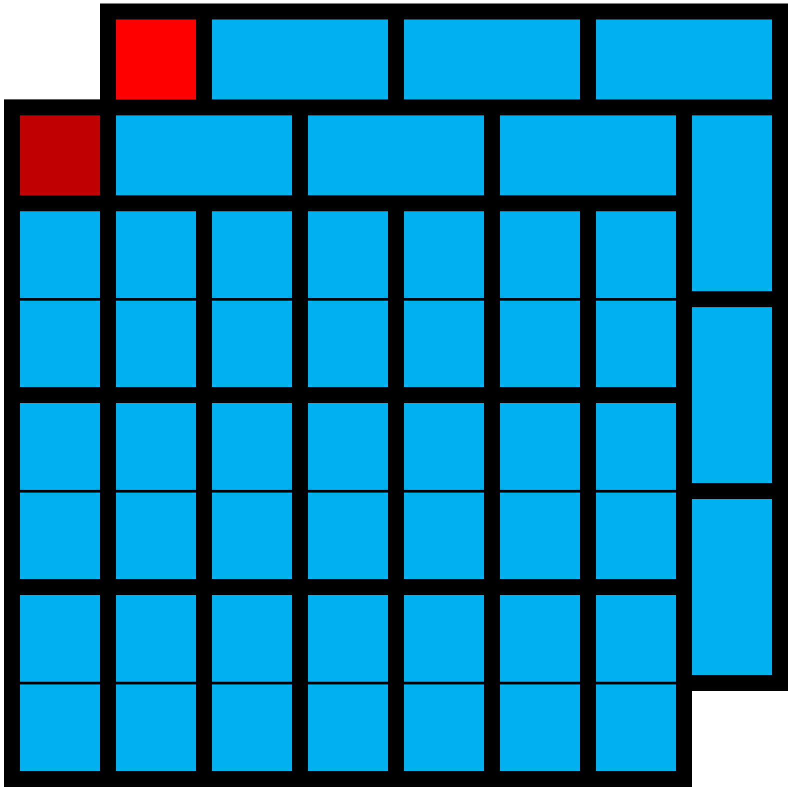
- Definitions, Domain, Codomain, Association,
- Examples that are not functions,
- injective, surjective, bijective

Functions in PYTHON

```
def f(x):
        return x + 3
```

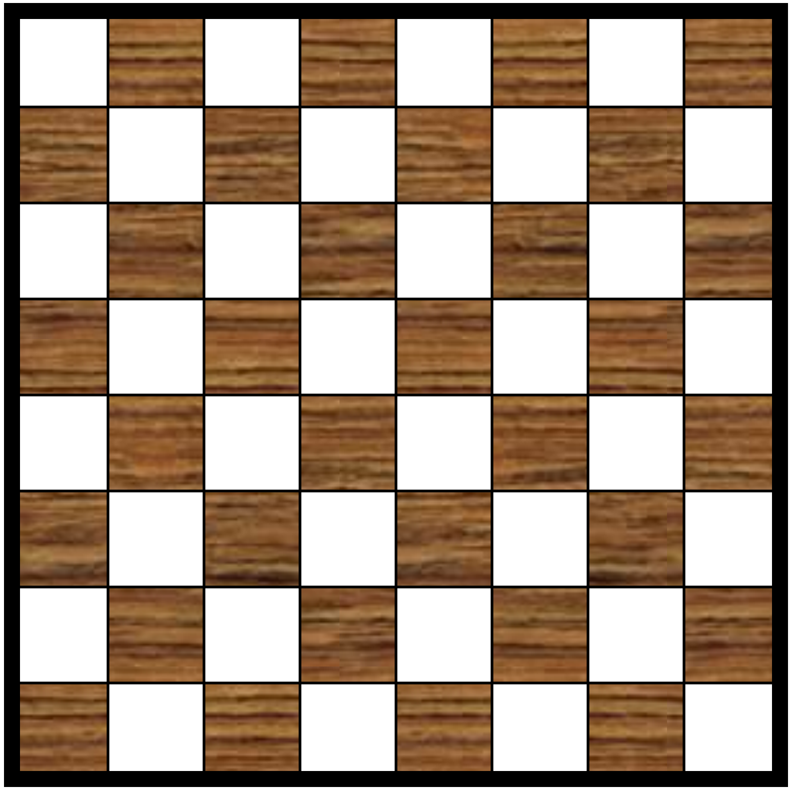
What did we do last time?

Demonstrations



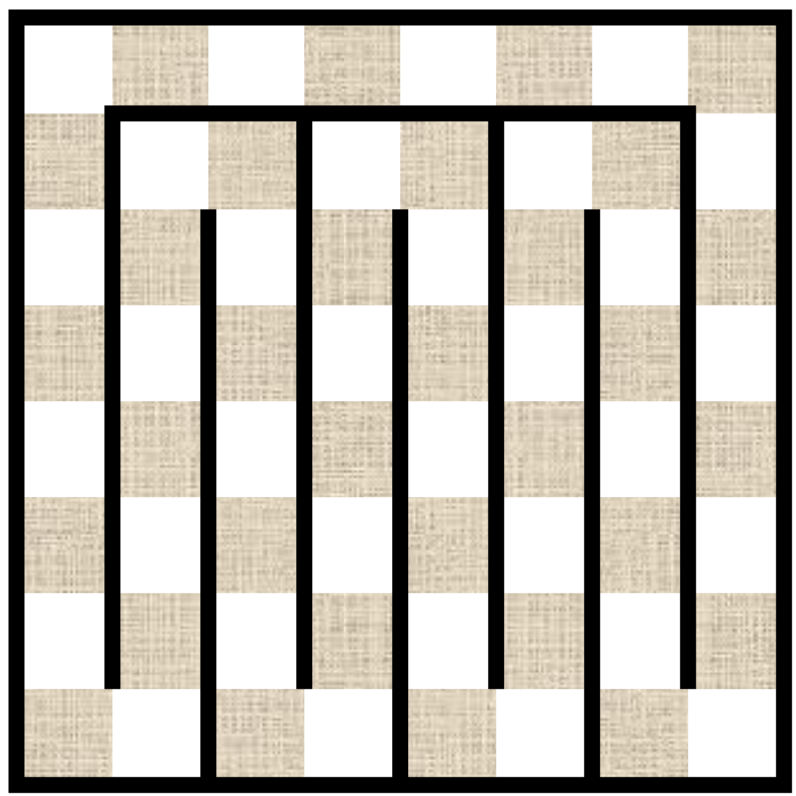
What did we do last time?

Can we prove that if we erase any pair of squares, one white and one black, we can still cover the board?



What did we do last time?

Can we prove that if we erase any pair of squares, one white and one black, we can still cover the board?



What did we do last time?

Demonstration by example

Proof by contradiction

A statement is equivalent to its contrapositive:

$$P \Rightarrow Q \quad \Leftrightarrow \quad \neg Q \Rightarrow \neg \mathrm{P}$$

the statemant — the contrapositive

Proof by mathematical induction

If a sentence P (n) depends on a natural number n and :

1) base case : P (1) is true

2) the inductive step: for any $n \geq 1$

$P(n) \Rightarrow P(n+1)$

then P(n) is true for any n .

What did we do last time?

Proof by mathematical induction

If a sentence P (n) depends on a natural number n and :

1) base case : P (1) is true
2) the inductive step: for any $n \geq 1$

   $P(n) \Rightarrow P(n + 1)$

then P(n) is true for any n .

-- --------------

What is the mathematical induction reasoning based on?

- P(1) proved to be true
- $P(1) \Rightarrow P(2)$, $P(2) \Rightarrow P(3)$, … , $P(n\text{-}1) \Rightarrow P(n)$, …

Complete Mathematical Induction

If a sentence $P(n)$ depends on a natural number n and :

1) base case : $P(1)$ is true

2) the inductive step: for any $n \geq 1$

$P(1)$ and $P(2)$ and $P(3)$ and ... and $P(n) \Rightarrow P(n+1)$ is true

then $P(n)$ is true for any n .

Complete Mathematical Induction

- Let's learn complete mathematical induction through a game!

Complete Mathematical Induction

Let's prove: Whatever strategy we choose to play, we will get the same score for the same number of pieces, S(n)

Ex: $S(8)=28$ $S(8)=56/2$ $S(8)=(7*8)/2$

$S(n)=((n-1)*n)/2$

$P(n)$

base case $P(1)=0$

Inductive step: $P(1)$ and $P(2)$ and $P(3)$ and ... and $P(n) \Rightarrow P(n+1)$

The score $S(n+1) = k*(n+1-k) + P(k) + P(n+1-k)$

It we need to prove that this score depends only on n, not k

Complete Mathematical Induction

$S(n+1) = k^*(n+1-k) + P(k) + P(n+1-k)$

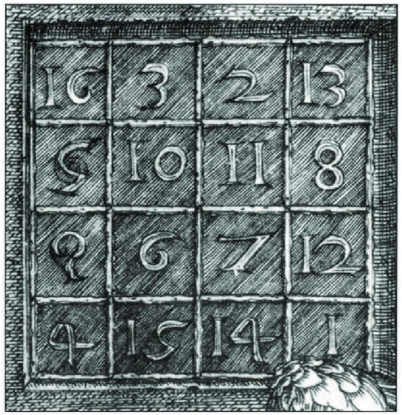
$S(n+1) = k^*(n+1-k) + \frac{k(k-1)}{2} + \frac{(n+1-k)(n-k)}{2}$

$S(n+1) = \frac{2kn+2k-2k^2+k^2-k+n^2-nk+n-k-nk+k^2}{2}$

$S(n+1) = \frac{(2kn-nk-nk)+(2k-k-k)+(-2k^2+k^2+k^2)+n^2+n}{2}$

$S(n+1) = \frac{n^2+n}{2}$

$S(n+1) = \frac{n(n+1)}{2}$



What did we do last time?

Complete Mathematical Induction

Sets, Tuples, Cartesian product

Functions – composition, invertible

Counting problems

Composition of functions in PYTHON

Inductively defined sets

The cardinality of a set

The cardinality of a set is the number of elements in the set.

The cardinality of a set is denoted as $|A|$.

We can have finite cardinalities: $|\{1, 2, 3, 4, 5\}| = 5$ or infinite cardinalities : N, R, etc.

What is the cardinality of an infinite set? $|N| = |R| = \infty$? Not. We have different cardinalities for infinite sets:

$|N| = \aleph 0$ – the smaller infinite

$|R| = 2^{\aleph 0}$

TUPLES

An *n-tuple* is a string of *n* elements ($x_1, x_2, \ldots, x_n$)

Properties :

- the elements are not necessarily distinct
- the order of the elements in the tuple matters

Special cases: *pair or twins* (*a, b*), *triple or triad* (*x, y, z*), etc.

Cartesian product

The *Cartesian product* of two sets is the set of pairs

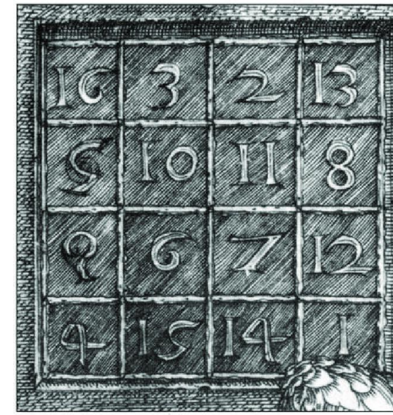
$A \times B = \{(a, b) \mid a \in A, b \in B \}$

The Cartesian product of n sets is the set of n – *tuples*

$A_1 \times A_2 \times \ldots \times A_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in A_i, 1 \leq i \leq n \}$

If the sets are finite, then

$| A_1 \times A_2 \times \ldots \times A_n | = | A_1 | \cdot | A_2 | \cdot \ldots \cdot | A_n |$



What did we do last time?

Complete Mathematical Induction

Sets, Tuples, Cartesian Product

Functions – composition, invertible

Counting Problems

Compiling Functions in PYTHON

Inductively defined sets

Function Composition

Let the *functions* $f : A \to B$ and $g : B \to C$.

Then the composition of f followed by g:

$$g \circ f : A \to C \quad (g \circ f)(x) = g(f(x))$$

We can compose $g \circ f$ only if the *codomain* of f = the *domain* of g !

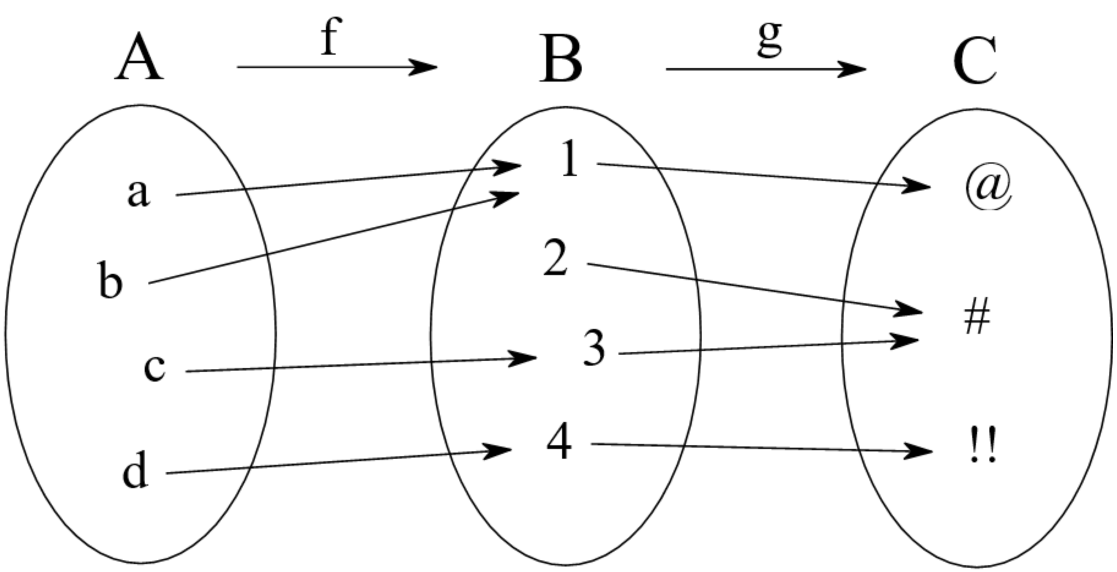


Image: http://en.wikipedia.org/wiki/File:Compufun.svg

Properties of Function Composition

- Function composition is associative:

$$(f \circ g) \circ h = f \circ (g \circ h)$$

- *Proof* :

$((f \circ g) \circ h)(x) =$
rewrite $\circ$ $= (f \circ g)(h (x))$
rewrite $\circ$ $= f (g (h (x)))$

$(f \circ (g \circ h))(x) =$
rewrite $\circ$ $= f ((g \circ h)(x))$
rewrite $\circ$ $= f (g (h (x)))$

- The composition is not necessarily commutative
- Can you give an example for which $f \circ g \neq g \circ f$?

Powers of Function

If f is a function on a set A, then the compositions $f \circ f$, $f \circ f \circ f$, . . . are valid, and we denote them as f^2, f^3, . . .

Definition:

Let $f : A \to A$

- $f^1 = f$; that is, $f^1(a) = f(a)$, for $a \in A$
- For $n \geq 1$, $f^{n+1} = f \circ f^n$; that is, $f^{n+1}(a) = f(f^n(a))$ for $a \in A$.

Invertible functions

On any set A we can define *the identity function:*

$ID_A: A \rightarrow A$, $id_A(x) = x$ (often noted $\mathbf{1}_A$ or $\mathbf{i}_A$)

A function $f : A \rightarrow B$ is *invertible* if it exists a function

$f^{-1} : B \rightarrow A$ such that

- $f^{-1} \circ f = id_A$ and
- $f \circ f^{-1} = id_B$. *(f^{-1}, read "f inverse")*

Invertible functions

A function is invertible if and only if it is bijective.

Bijections have inverses.

Let $f : A \rightarrow A$. f^{-1} exists if and only if f is a bijection; f is *one-to-one* and *onto.*

What did we do last time?

Complete Mathematical Induction

Sets, Tuples, Cartesian Product

Functions - Composition, Invertible

Counting Problems

Compiling functions in PYTHON

Inductively defined sets

How many functions are there from A to B?

If A and B are finite sets, there are $|B|^{|A|}$ functions from A to B . (every element of B can be mapped to any element of A)

Proof : by *mathematical induction* after $|A|$

The set of functions $f : A \to B$ is sometimes denoted by B^A

The notation reminds us that the number of these functions is $|B|^{|A|}$

How many injective functions are there from A to B?

If A and B are finite sets and $f : A \to B$ is injective $\Rightarrow |f(A)| = |A|$ (the image of f will have $|A|$ elements)

The order in which we choose the *elements matters*! (different orders $\Rightarrow$ different functions)

... so we have arrangements of | B | taken as | A |

$\Rightarrow$ exists $A_{|B|}^{|A|} = \frac{|B|!}{(|B|-|A|)!}$ injective functions

How many bijective functions does A to B exist?

If A and B are finite sets and $f : A \rightarrow B$ *is* bijective

$\Rightarrow |f(A)| = |A| = |B|$ (the image of f will have $|A|$ elements).

The order in which we choose the *elements matters*!

... so we have permutations of $|A|$ element

$\Rightarrow$ exists $P/A/ = |A|!$ bijective functions

What did we do last time?

Complete Mathematical Induction

Sets, Tuples, Cartesian Product

Functions - Composition, Invertible

Counting Problems

Composition of Functions in PYTHON

Inductively defined sets

Types of data

Python provides 4 types of primitive data :

- Integer
- Float
- String
- Boolean

The primitive data types in Python are immutable. This means that once they are created, their values cannot be changed.

If you assign a new value to a variable of a primitive data type, a new object is created with the updated value, rather than modifying the original object.

Integer

Integers represent whole numbers without decimal points.

They can be positive or negative, and there is no limit to their size.

Ex: 3, 6, -234.

Float

Floats, or floating-point numbers, represent numbers with decimal points.

They can also be positive or negative and can have a fractional part.

Ex : 3.34, -0.123456.

Float

2 is an integer value. For a real value (float) we must write 2.0 (or abbreviated 2.).

In Python the type conversion from int to float is done automatically. Thus, the result of operations containing both integers and real numbers will be a real number (e.g. 5 + 2.0 will give 7.0).

We can also use the float() function if we want to do a conversion explicitly:

```
>>> float (3 * 2)
6.0
```

String

Strings are sequences of characters enclosed in single or double quotation marks.

They are used to represent text and can contain letters, numbers, symbols, and spaces.

Example : 'book', "23abc ".

Concatenation is a common operation when working with strings, and it allows us to build longer strings by combining smaller ones. This is done by the + operator :

>>> ' abc ' + ' def '

' abcdef '

String

The characters from a string can be accessed via "string"[index]:

```
>>> 'A string'[2]
's'
```

```
>>> 'A string'[-8]
'A'
```

The result is the third character (character numbering starts from 0). Python also allows accessing characters using negative index values.

For the example below, selecting any other integer that is outside the range [-8; 7] will generate an exception

'A '	' '	's '	't '	'r'	'i'	'n'	'g'
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

Boolean

Booleans are a special type that can have one of two values: True or False.

They are often used in logical operations and conditional statements.

```
>>> 3 == 4
   False
```

Using comparison operators == (equal), ! = (different), > (bigger), < (smaller), > = (bigger or equa), < = (smaller or equal), we can make comparisons between different expressions.

To write more complicated conditions we can use the keywords and, or and not.

Predefined types for data collections

In Python, there are several predefined types for data collections.

These types allow us to store and manipulate collections of data in a structured manner.

The four main predefined types for data collections in Python are:

- List
- Tuple
- Set
- Dictionary

List

A list is an ordered collection of elements, enclosed in square brackets ([]).

It can contain elements of different types and allows for duplicate values.

Lists are mutable, meaning that we can modify their elements.

even = [0, 2, 4, 6, 8]

If we want to access a list item we do the same as for strings

Example : if we want to access the second item in the even list we write even[1], in this case also indexing from 0.

Tuple

A tuple is similar to a list, but it is enclosed in parentheses (()).

Tuples are also ordered collections, but unlike lists, they are immutable, meaning that their elements cannot be modified once defined.

tuple_even_numbers = (0, 2, 4, 6, 8)

Set

A set is an unordered collection of unique elements, enclosed in curly braces ({}).

Sets do not allow duplicate values, and they are mutable, meaning that we can add or remove elements from them.

a_set = {6, 0, 2, 4, 8}

Dictionary

A dictionary is a collection of key-value pairs, enclosed in curly braces ({}).

Each element in a dictionary consists of a key and its corresponding value.

Dictionaries are mutable and allow for efficient lookup of values based on their keys.

dict= {1:”one”, 2:”two”}

Function Composition

- The result of function f becomes the argument to function g
- By composition, we construct complex functions from simpler functions.

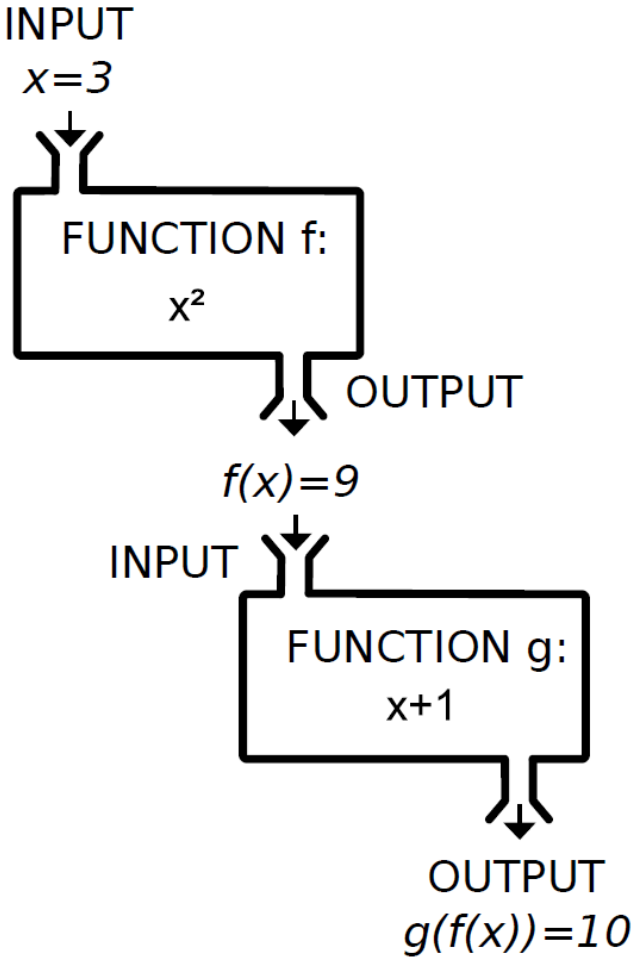


Image: http://en.wikipedia.org/wiki/File:Function_machine5.svg

Function Composition in PYTHON

Composing functions is the way in which the result produced by one function is used as a parameter in another function.

If we have two functions, f and g, their composition is represented as: f(g(x)), where x is the argument to function g, and the result of function g(x) becomes the argument to function f

Function Composition in PYTHON

```
def sum (x):
        return x + 10

def product (x):
        return x * 10

print(product ( sum (5)) )             # (5+10)*10
150
```

Function Composition in PYTHON

We can also create a new function that combines 2 existing functions:

```
def sum (x):
        return x + 10
def product (x):
        return x * 10

def compound_function (f, g):
        return lambda x : f(g(x))

product_sum = compound_function(product, sum)
print (product_sum(2))
```

Composition of 3 functions in Python

```
def sum (x):
        return x + 10
def product (x):
        return x * 10
def difference (x):
        return x - 2

def compound_function (f, g):
        return lambda x : f(g(x ))
product_difference_sum = compound_function(product,
compound_function(difference, sum ))
print (sum_difference_product(2))
```

A function can be returned by another function in PYTHON

```
# Functions can return another function

def create_adder (x ):
        def adder(y):
                return x+y
        return adder

add_15 = create_adder (15 )
print (add_15( 10 ))
```

OUTPUT:
25

Example retrieved from : https://www.geeksforgeeks.org/first-class-functions-python/?ref=lbp

Functions - extra information

We can write the value of the arguments and the result in code:

```
def sum( a : int , b : int ) -> int :
        c = a + b
        return c
```

Functions - extra information

We can write functions with predefined arguments:

```
def sum (x , y= 10 ):
        return x+y
```

We can call the function with only one parameter:

sum (8)	sum(1,2)

OUTPUT:

18	3

Functions - extra information

When calling, we can write the parameters in a different order than in the function definition:

```
def difference ( n1 , n2 ):
    return n1-n2
```

We can call the function:

```
difference ( n1= 10 , n2= 3 )
difference (n2= 3 , n1= 10 )
```

OUTPUTS:

7

7

Summary of functions

By functions we express calculations in programming.

The definition fields and values correspond to the types in programming.

In functional languages, functions can be manipulated like any values. Functions can be arguments and results of functions.

What do we know so far?/ What should we know?

- We know the properties of functions and how to use them: injective, surjective, bijective, invertible functions;
- We know how to construct functions with certain properties;
- We know how to count functions defined on finite multitudes (with given properties);
- We know how to compose simple functions to solve problems;
- We know how to identify the type of a function.

Thank you!

Bibliography

- The full math induction game was inspired by ***the Mathemati cs for Computer Science course*** from Massachusetts Institute of Technology (from https://ocw.mit.edu /)
- The content of the course is mainly based on the materials of the past years from the LSD course, taught by Prof. Dr. Marius Minea et al. Dr. Eng. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)