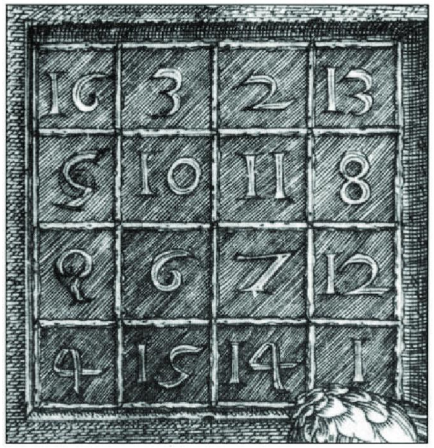
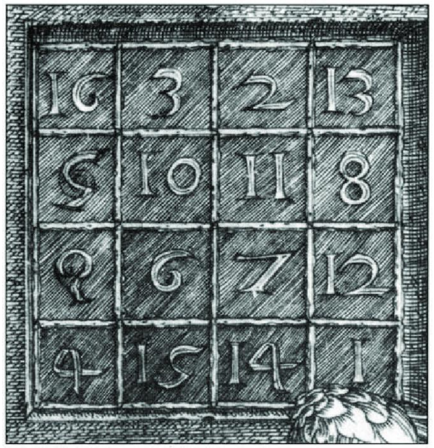
Logic and Discrete Structures -LDS



Course 5 – Sets

Lecturer Dr. Eng. Cătălin Iapă

catalin.iapa@cs.upt.ro

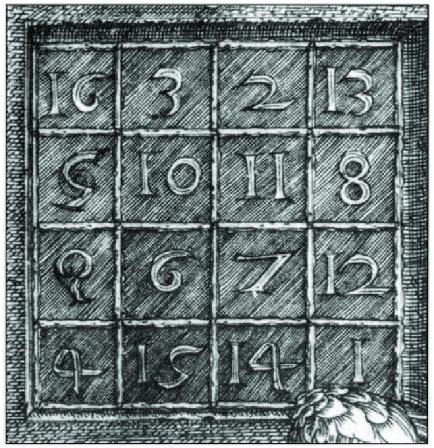


What have we covered so far?

Functions

Recursive functions

Lists



What are sets?

Basic set operations

Sets, foundation of mathematics

Sets in PYTHON

Set Operations in PYTHON

Exercises with sets in PYTHON

What are sets?

Set is a *fundamental mathematical concept* .

We could say:

Definition: A set is a collection of objects called the elements of the set.

But to be rigorous, we should define precisely what a collection is.

The definition is informal. We'll see that it's important to formalize it.

What are sets?

We have two distinct notions : *element and set*

$x \in S$: the element x is a member of the set S

$y \notin S$: the element y is not a member of the set S

Unlike lists:

The order of the elements does not matter

$\{1, 2, 3\} = \{2, 1, 3\}$

An element cannot appear more than once

$\{1, 2, 3, 2\}$

How do we define sets?

By *description* : { the set of divisors of 6 }

By *listing* elements:
$A = \{ a, b, c \}$, $D = \{1, 2, 3, 6\}$ = the set of divisors of 6
The elements of the set are written between braces, separated by a comma.

By a characteristic property:
$S = \{ x \mid x$ has the property $P(x)\}$
$D(n) = \{ d \in \mathrm{N} \mid n \bmod d = 0\}$ (the set of divisors of n)

We know : the set of natural numbers N, integers Z, rationals Q, reals R, ...

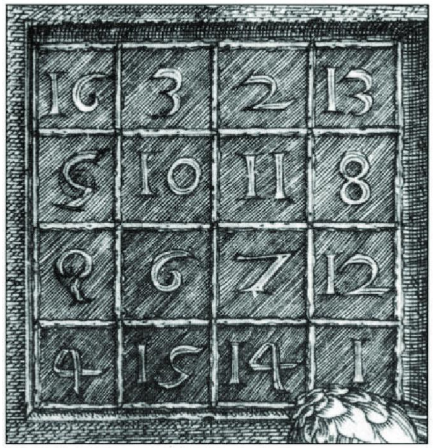
Subsets

A is *a subset* of B : $A \subseteq B$
if every element of A is also an element of B .
A is a *proper subset* of B : $A \subset B$
if $A \subseteq B$ and there is (at least) an element $x \in B$ such that $x \notin A$.

$\in$ is a relationship between an *element* and a lot .
$\subseteq$ and $\subset$ *are* relations between *two sets* .

To *prove* that $A \nsubseteq B$ it is enough to find an element $x \in A$ for which $x \notin B$.

If $A \subseteq B$ and $B \subseteq A$, then $A = B$ (the sets are equal)

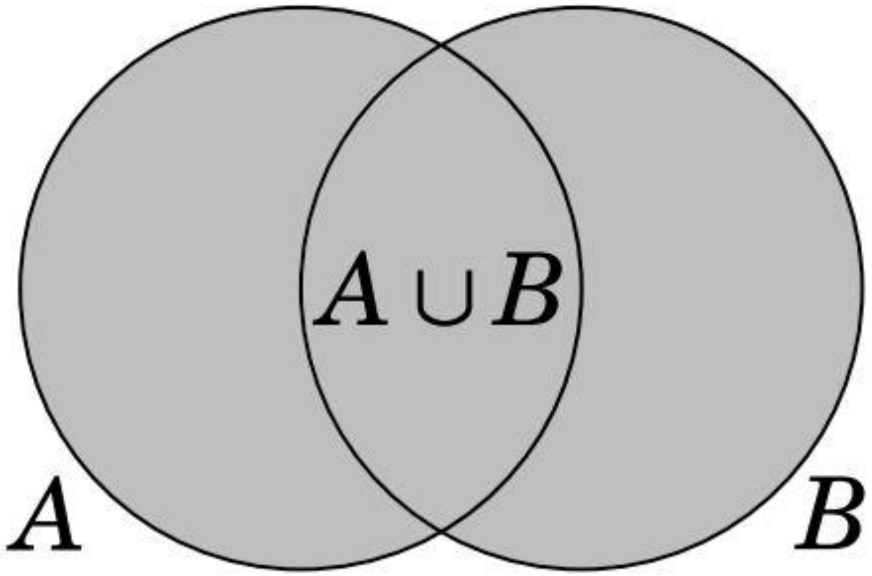

16 3 2 13
5 10 11 8
9 6 7 12
4 15 14 1

Basic set operations

The union of sets :

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

Venn diagram representation:

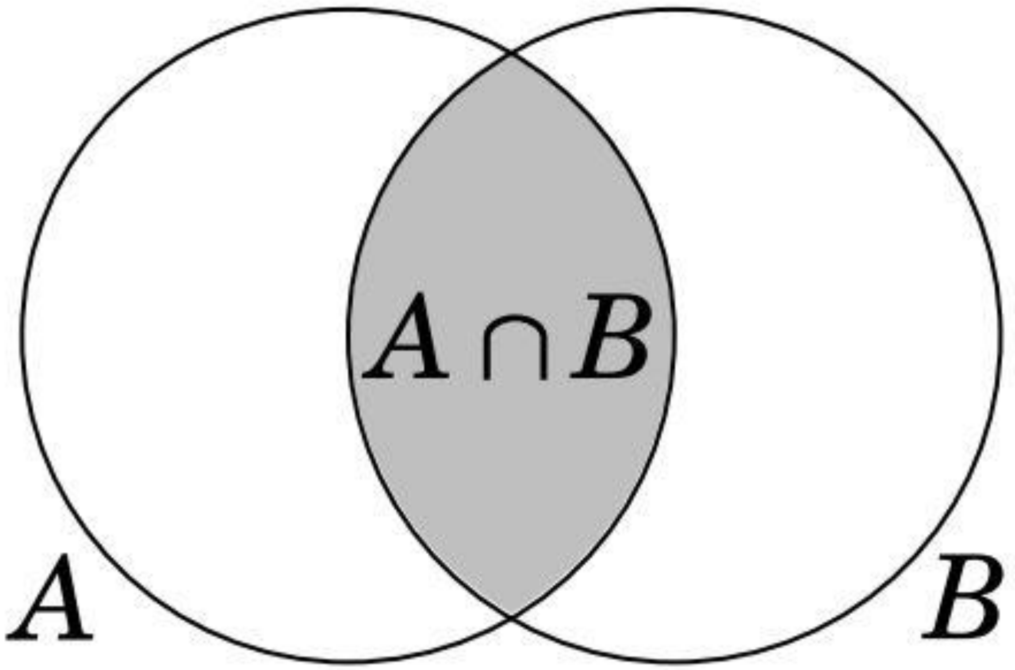


https://en.wikipedia.org/wiki/Venn_diagram

Basic set operations

The intersection of sets:

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

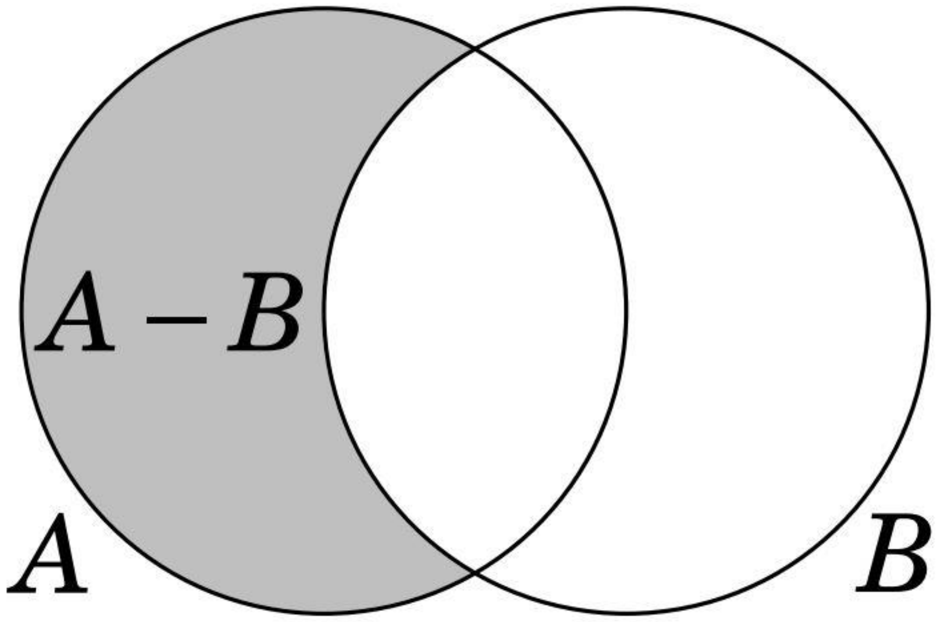


https://en.wikipedia.org/wiki/Venn_diagram

Basic set operations

The difference of sets:

$$A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$$



https://en.wikipedia.org/wiki/Venn_diagram

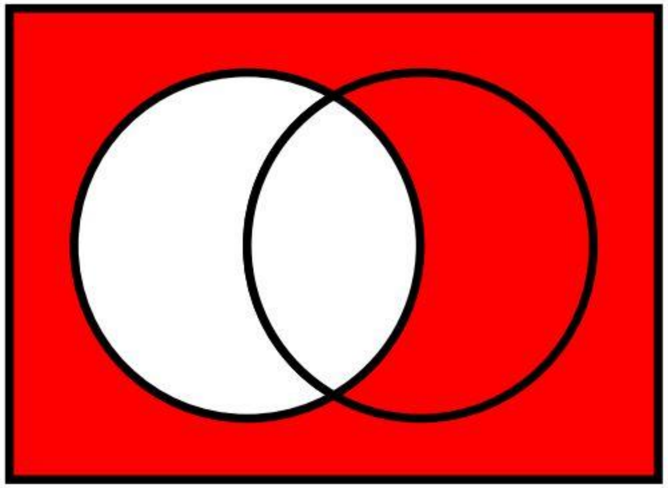
Basic set operations

Usually, we discuss it in a context: we have a universe U of all the elements we could refer to.

The complement of a set (with respect to the universe U):

A^C = {$x \in U$ | $x \notin A$ } = U \ A (*denoted also* $\bar{A}$)

.



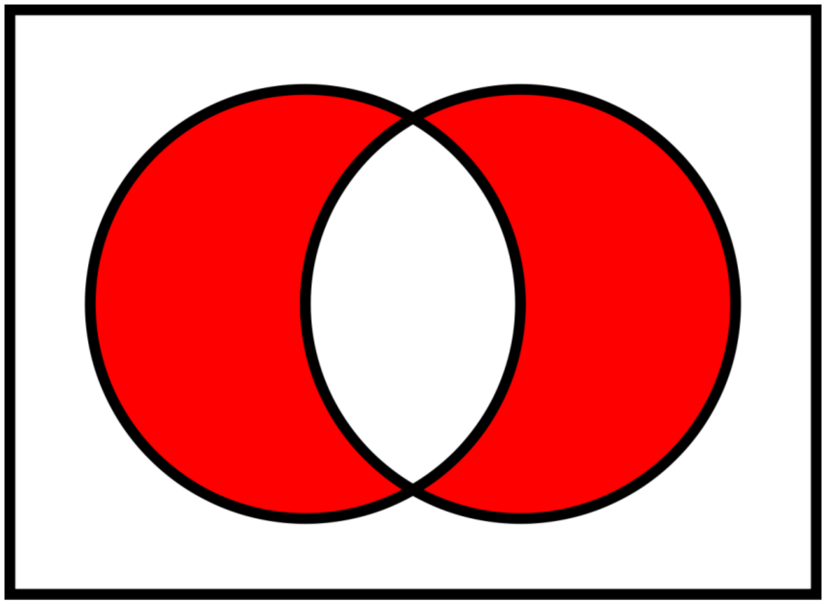
https://en.wikipedia.org/wiki/Venn_diagram

Basic set operations

Symmetric difference of sets:

$$A \Delta B = (A \setminus B) \cup (B \setminus A)$$

.



https://en.wikipedia.org/wiki/Venn_diagram

Basic set operations

If we fix the universe U of elements, we can represent any set $S \subseteq U$ by the *characteristic function*

$$f_S : U \rightarrow \mathrm{B} : f(x) = \begin{cases} \textit{True} \text{ if } x \in S \\ \textit{False} \text{ else (if } x \notin S) \end{cases}$$

.

Boolean algebra of sets

The notion is due to the mathematician George Boole (19th century)

The operations of a Boolean algebra (here ∪ and ∩) satisfy the properties:

Commutativity : $A \cup B = B \cup A$ $\qquad A \cap B = B \cap A$

Associativity :

$(A \cup B) \cup C = A \cup (B \cup C)$ and $(A \cap B) \cap C = A \cap (B \cap C)$

Distributivity : $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ and

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Boolean algebra of sets

The operations of a Boolean algebra (here ∪ and ∩) satisfy the properties:

Identity : there are two values (here ∅ and the universe U) such that:

$$A \cup \emptyset = A \qquad\qquad A \cap U = A$$

Complement : every A has a complement A^c (or $\bar{A}$) such that:

$$A \cup A^c = U \qquad\qquad A \cap A^c = \emptyset$$

Boolean algebra of sets

Other properties (can be deduced from the above):

Idempotence : $A \cup A = A$ $A \cap A = A$

Absorption : $A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$

Double complement : $(A^c)^c = A$

Boolean algebra of sets

Other properties (can be deduced from the above):

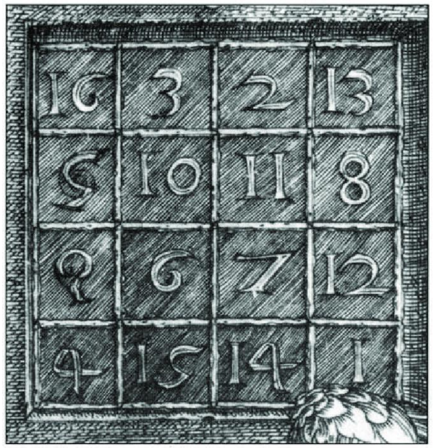
Complements of identity elements : $\emptyset^c = U$ $U^c = \emptyset$

Universal limit : $A \cup U = U$ $A \cap \emptyset = \emptyset$

De Morgan's laws :

$$(A \cup B)^c = A^c \cap B^c \qquad (A \cap B)^c = A^c \cup B^c$$

We will review these laws in *propositional logic* .



What are sets?

Basic set operations

Sets, foundation of mathematics

Sets in PYTHON

Set Operations in PYTHON

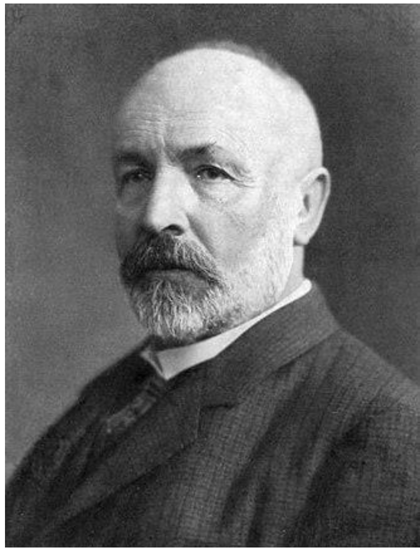
Exercises with sets in PYTHON

Sets, foundation of mathematics

The set is one of the most *important concepts* in modern mathematics .

Georg Cantor (1874) - created set theory, which became *a fundamental theory of mathematics* in the work "Uber eine Eigenschaft des Inbegriffes aller reellen algebraic Zahlen ” (“On a Property of the Collection of All Real Algebraic Numbers").

Known as *Naive Set Theory*.



https://ro.wikipedia.org/wiki/Georg_Cantor

Sets, foundation of mathematics

Cantor established the importance of *one-to-one correspondences* between the members of two sets, defined *infinite* and well-*ordered sets* , and showed that the real numbers are much *more numerous* than the natural numbers.

Cantor's method of proof of this theorem implies the existence of an infinity of infinities.

He defined the cardinal and ordinal numbers and their arithmetic.

Cantor's work is of great philosophical interest.

https://ro.wikipedia.org/wiki/Georg_Cantor

Sets, foundation of mathematics

Practically all mathematics can be formalized in set theory.
(or in logic, to which it is closely related, as we shall see)

Example: *a pair* (ordered) can be defined:
(a, b) = {{a} , {a , b}} - how can we extract *a* and b from (*a, b*)?
Intersection , difference
(1921, Kazimierz Kuratowski - Polish mathematician who made important contributions to topology, set theory and graph theory.)

But starting from imprecise, natural language definitions, paradoxes appear in the naive theory of sets.

https://ro.wikipedia.org/wiki/Georg_Cantor

Russell's paradox

Let R *be* the set of all sets that do not contain themselves:

$$R = \{ X \mid X \notin X \}$$

Does the set R contain itself?

- if yes $R \in R$, to satisfy the definition condition , we have $R \notin R$.
- if $R \notin R$, then R satisfies the condition , so $R \in R$: *paradox* !

An intuitive formulation (the barber's paradox):

- The barber shaves exactly the people who don't shave themselves. Does the barber shave himself or not?

Russell's paradox

The paradox caused serious problems for the formalization of mathematical logic.

It can be *avoided* in several ways , imposing *restrictions* on how a set can be defined .

e.g.: We cannot define a set only by a property $P(x)$, we must *specify the universe* from which it can take its elements:

$$R = \{ X \mid X \subseteq U \text{ and } X \notin X \}$$

Axiomatic set theory

An axiom is a sentence that is assumed to be true. It is a starting point for reasoning.

Axiomatic systems were developed to avoid paradoxes of naive *set* theory *(* with notions defined in natural language)

The most widespread : *the Zermelo-Fraenkel system* (developed between 1907-1930).

Axiomatic set theory

A few axioms :

Axiom of extensionality :

Two sets are equal if and only if they have the same elements

(if every element of A is also an element of B , and vice versa)

$$\forall A, \forall B\ (A = B \Leftrightarrow \forall c\ (c \in A \Leftrightarrow c \in B))$$

Axiom of the empty set (existence):

There is a set that has no element

$$\exists E \forall X\ \neg(X \in E\ )$$

Axiomatic set theory

Axiom of regularity (of foundation)

Every nonempty set has an element $x \in A$ disjoint from it:

$x \cap A = \emptyset$

$$\forall X\,(X \neq \emptyset) \Rightarrow \exists Y\,(Y \in X \;\wedge\; \neg\exists Z\,(Z \in X \;\wedge\; Z \in Y\,))$$

It follows that there is no infinite sequence $A_0, A_1, \ldots A_n \ldots$ so that

$$A_0 \ni A_1 \ni \ldots \ni A_n \ni \ldots$$

($\{ A_0, A_1, \ldots \}$ would be such a set)
It follows that *no set can have as element* , $X \notin X$,

otherwise $X \ni X \ni X \ldots$ would be such a string

Intuitive: any set is made up of simpler elements (possibly sets), which in turn contain simpler elements, until we reach *fundamental elements*

⇒ that *eliminates* Russell's paradox

Cardinal of a set

The cardinal (cardinality) of a set is the number of elements of the set.

Cardinal of a set is noted |A|.

Can have sets:

- *finite* : |{1, 2, 3, 4, 5}| = 5 or
- *infinite* : N, R, etc.

Cardinal of reunion, intersection, difference

For *finite* sets :

Reunion law:

$$| A \cup B | = | A | + | B | - | A \cap B |$$

Difference law:

$$| A \setminus B | = | A | - | A \cap B |$$

The power set (the set of sudsets)

The set of subsets (*power set*) of a set *S*, denoted P(*S*) (sometimes 2^S):

$$P(S) = \{ X \mid X \subseteq S \}$$

Example, for $S = \{ a, b, c \}$, we have:

$P(S) = \{\emptyset , \{ a \}, \{ b \}, \{ c \}, \{ a, b \}, \{ a, c \}, \{ b , c \}, \{ a , b, c \}\}$

If *S* is finite, then $|P(S)| = 2^{|S|}$

Tuples and Cartesian product

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* (a, b), *triplet* (x, y, z)

Tuples and Cartesian product

Cartesian product of two sets is the set of pairs

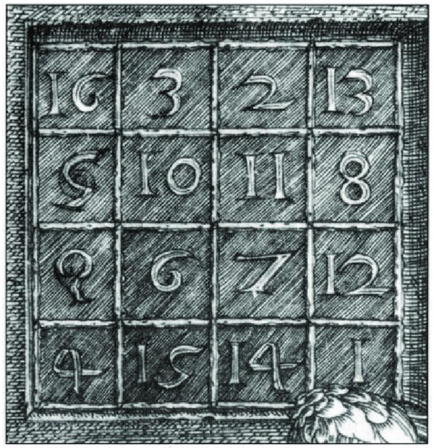
$A \times B = \{(a, b) \mid a \in A, b \in B\}$

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 are sets?

Basic set operations

Sets, foundation of mathematics

Sets in PYTHON

Set Operations in PYTHON

Exercises with sets in PYTHON

Sets in PYTHON

Sets are collections that hold multiple elements in a single variable.

They are one of *the basic collections* in PYTHON (besides Lists, Tuples and Dictionaries)

Sets are collections:

- *unordered*
- *not indexed*
- *do not allow duplicates* among elements
- must contain only immutable elements

Sets in PYTHON

To *create* an array we list the elements of the array in *curly braces* { } or we will use the constructor *set()*

```
set1 = {1, 2, 3, 4, 5}
set2 = set ((1, 2, 3, 4, 5))
set3 = set([1, 2, 3, 4, 5])
```

Sets in PYTHON

To *create an empty set* we will only be able to use the *set() function*, if we define an empty set with two braces, PYTHON will interpret it as a dictionary.

```
X = set()
print ( type ( X ) )
# < class 'set'>

Y = {}
print ( type ( Y ) )
# < class ' dict '>
```

Sets in PYTHON

The order the elements do not matter. On display, the elements may appear in *a different order*.

```
set1 = {1, 11, 4, 5, 3, 2}
print ( set1 )

# {1, 2, 3, 4, 5, 11}
# {1, 2, 3, 4, 11, 5 }
# { 1, 3, 4 , 2 , 11, 5 }
```

Sets in PYTHON

Duplicate items will be retained only *once*

```
set1 = {1, 11, 4, 5, 3, 2 , 1, 1, 2 }
print ( set1 )

# {1, 2, 3, 4, 5, 11}
```

Sets in PYTHON

To find the number of elements in a set we can use the *len() function*

```
set1 = {1, 11, 4, 5, 3, 2}
print ( len ( set1 ) )

# 6
```

Sets in PYTHON

The elements can have *different types of data* :

days = {“ monday ”, “ tuesday ”, “ wednesday ”}
numbers = {1, 2, 3}
values = {True, False}

A set can contain different types of data *at the same time*:

set1 = {“ monday ”, 1 , True }

Sets in PYTHON

The elements of a set are immutable.

A tuple can be an element of a set:

x = { 3 , 4 , (1, 2, 3)}

A list (or a dictionary) cannot be an element:

A={3, [4, 5,6]}

It will generate error

TypeError : unhashable type: 'list'

Access to set elements

Set elements *cannot be accessed by index*.

We can check if an element is in a set with *in*:

```
if (x in {1, 2, 3 } )
        print ("item is in the set")
else
        print ("element is not in the set")
```

Adding new items

Once a set is created, its elements cannot be modified, instead elements *can be added* or *deleted* from the set.

We can add new elements to the set with the *add() method*

```
set1 = {1, 11, 4, 5, 3, 2 }
set1.add (29)
print (set1)
# {1, 2, 3, 4, 5, 11, 29}
```

Adding new items

We can add *all the elements of another set* to the current set with the *update() method*

```
days = {"Monday","Tuesday","Wednesday"}
weekend_days = {“Saturday",“Sunday"}
days.update (weekend_days)
print (days)

#{ 'Sunday', 'Tuesday', 'Wednesday', 'Monday',
'Saturday'}
```

Deleting items

We can delete elements from a set using remove() or discard() methods

```
days = {"Monday","Tuesday","Wednesday"}
days.remove ( "Tuesday" )
days.discard ( "Wednesday" )
print (days )

# { 'Monday'}
```

Deleting items

remove() method will throw an error if called with a non-existent element, while the *discard() method* will not throw an error.

```
days = {"Monday","Tuesday","Wednesday"}
days.remove ("Thursday") # will generate error
days.discard ("Thursday") #will not generate error
print (days)
```

Deleting items

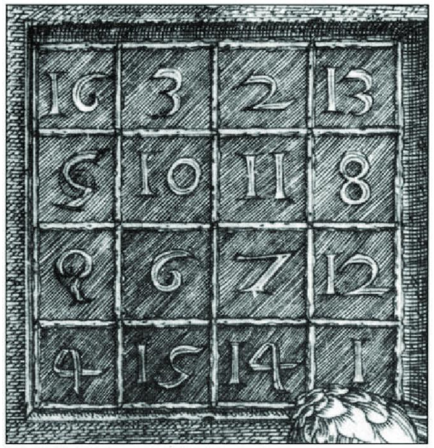
To delete all elements of a set we use the *clear() method*

```
days = {"Monday","Tuesday","Wednesday"}
days.clear()
print (days)

# set()
```

Deleting items

To delete the set completely we can use del

```
days = {"Monday","Tuesday","Wednesday"}
del days
print (days)

# print( days )
# NameError : name ' days' is not defined
```



What are sets?

Basic set operations

Sets, foundation of mathematics

Sets in PYTHON

Set Operations in PYTHON

Exercises with sets in PYTHON

Sets Operations

We can compute the reunion of two sets with the *union() method* . The union() method will create *a new set* that will contain the elements of both sets.

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1. union (set2)
print (set3 )

# { 'a', 1, 'b', 2, 3, 'c '}
```

Sets Operations

We can calculate the intersection of two sets with the *intersection() method* .

```
set1 = { 2, 3, 4, 5 }
set2 = {1, 2, 3}
set3 = set1. intersection (set2)
print (set3)

# {2, 3}
```

Sets Operations

We can calculate the difference of two sets with the *difference() method* .

```
set1 = { 2, 3, 4, 5 }
set2 = {1, 2, 3}
set3 = set1.difference(set2)
print (set3)

# {4, 5}
```

Sets Operations

We can calculate the symmetric difference of two sets with the *symmetric_difference() method*.

```
set1 = { 2, 3, 4, 5 }
set2 = {1, 2, 3}
set3 = set1. symmetric_difference (set2)
print (set3)

# {1, 4, 5}
```

Sets Operations

union(), intersection (), difference() methods can also be used with multiple arguments:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
C = {3, 4, 5, 6}
D = {4, 5, 6, 7}
E = A.union (B, C, D)             # E = {1, 2, 3, 4, 5, 6, 7 }
F = A.intersection (B, C)         # F = {3, 4}
G = A.difference (C, D)           # G = {1, 2 }
```

Method *symmetric_difference()* has only one argument .

Sets Operations

We can check whether a set is a subset or superset of another set with the methods issubset() and issuperset()

```
A = {1, 2, 3, 4}
B = {2, 3}

print(B.issubset(A))                         # True
print(A.issubset(B))                         # False

print(B.issuperset ({1, 2, 3, 4}) )          # False
print(A.issuperset ({1, 2, 3, 4} ))          # True

print(A.issubset(A))                         # True
```

Sets Operations

In Python we can use *operators for sets*:

Reunion	the operator |	C = A| B
Intersection	the operator &	D = A & B
Difference	the operator –	E = A - B
The symmetric difference	the operator ^	F = A ^ B
Subset	the operators < and <=	A < B
Superset	the operators > and >=	B >= A

Sets Operations

Example :

```
A = {1, 2, 3, 4}
B = { 3, 4, "a", "b", "c "}
C = A | B                    #C = {1, 2, 3, 4, 'a', 'b', 'c '}
D = A & B                    #D = {3, 4 }
E = A – B                    #E = {1, 2 }
F = A ^ B                    #F = {1, 2, 'a', 'b', 'c'}
```

Sets Operations

If we want the result of an operation to be found in *the current set* and not to generate *a new set* with the result of the operation, we will use the methods: *update()*, *intersection_update ()* , *difference_update ()*, *symmetric_difference_update ()*:

A = {1, 2, 3, 4}

B = {2, 3, 4, 5}

A.update(B) *# A = {1, 2, 3, 4, 5}*

A.intersection_update(B) *# A = {2, 3 , 4, 5}*

A.difference _update(C) *# A = {2}*

Note that each operation above will modify the set A, and the next method will use *the new composition* of A

Sets Operations

If we want a set to have *another set as an element* and we write:

A = {{1,2,3}, {7}}

We will receive error *TypeError : unhashable type: 'set'*

It does not let us write this syntax because the elements of the set A must be *immutable objects.*

To be able to do such operations, PYTHON provides us with a set that does not allow changes once it has been created: frozenset

Sets Operations

Sets of type *frozenset* can use all the methods and operators of the set data type *except* those that modify the structure of the set.

Example:

```
A = frozenset ( { 'a ', 'b', 'c ' } )
print (A)    # frozenset ({'a', 'b', 'c'})
print (len(A)) # 3
print (A & {'a', 'b','z '})   # frozenset ({'a', 'b '})
A = { frozenset ({1,2,3}), frozenset ({7})}

print (A. add ('d '))                    # this call will generate the error :
AttributeError : ' frozenset ' object has no attribute 'add'
```

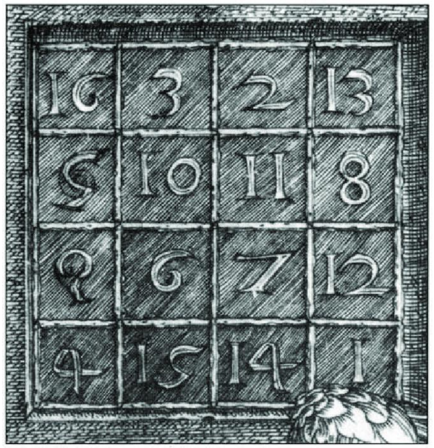
Sets Operations

The *map(), filter(), and reduce()* functions discussed for working with lists can be used similarly for sets. The 3 functions have an *iterable* as parameter.

We can iterate over a set (in the style of functional programming) with the help of the *reduce() function:*

```
import functools
M = {1, 2, 3, 4}
functools.reduce(lambda acc, elem: print(elem), M, 0)
```

The last argument of the reduce function (0, in the example above) indicates from which *initial value* to loop the function.



What are sets?

Basic set operations

Sets, foundation of mathematics

Sets in PYTHON

Set Operations in PYTHON

Exercises with sets in PYTHON

Exercises with sets

1. Write a function that returns the set of all divisors of a positive number n given as an argument. Iterate using recursive functions.

```
def set_of_divisors (n, A= set (), i =2):
    if ( i >n/2):
        return A
    else :
        if (n % i == 0):
            A. add ( i )
        return set_of_divisors (n, A, i+1)

print ( set_of_divisors (20))
```

Exercises with sets

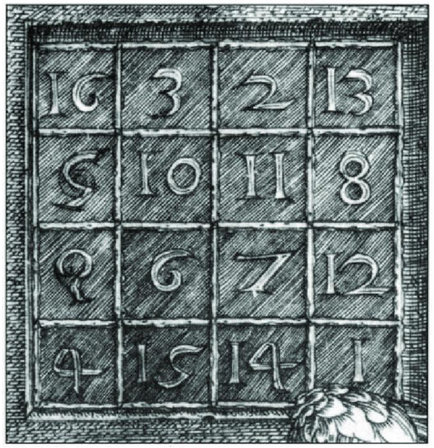
2 . Implement the standard filter function that takes as parameters a Boolean function (condition, predicate) f and a set s and returns the set of elements in s that satisfy the function f. Iterate the original set with the reduce() function.

Below is an example of using the *filter()* function , which we need to implement ourselves:

```
def odd (x):
        return x % 2

B= set ( filter ( odd , {1, 2, 3, 4, 5, 6}))
print (B )
# {1, 3, 5}
```

Exercises with sets

```
import functools
def my_filter (f , A, B= set ()):
    def function ( acc , elem ):
        if (f( elem )):
            B.add ( element )
        return f( element )
    functools.reduce ( function , A, 0 )
    return B

def odd (x):
    return x % 2
B=set( my_filter ( odd , {1, 2, 3, 4, 5, 6}))
print (B )
# {1, 3, 5}
```



Thank you!

Bibliography

- 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)