**Fundamentos de Programação: Python**

Slide 1:

Python:

* Interpreted language- line by line
* Slower than compiled languages
* Basic data structures and small utility functions are built in
* Hundreds of extension libraries
* Two Modes: Interactive and script

Interactive Mode:

1. >>>, prompt, indicates it is ready
2. When you type an expression, the interpreter prints the result

Script Mode

1. You can store in file, ending in “.py” and use the interpreter to execute it 🡪 Script/Program
2. In script mode does NOT show prompt
3. Indentation is crucial, same amount
4. Statements typically end with a new line, “\” indicates continues in the next line
5. Semicolon (;) allows multiples statements on a single line

Program

A sequence of statements that specifies how to perform a computation

* Assigment: store values in variables
* Math: basic math operations
* Input: get data
* Output: display data or send data
* Conditional execution: check conditions and execute appropriate code
* Repetition: perform actions repeatedly

Errors and Debugging

* Programming errors🡪 bugs
* Tracking down and correcting bugs🡪debugging
* Three kinds of errors:
* Syntax- not respect syntactical rules
* Runtime- only appears after program starts running, caused by type mismatches, or failure to deal with special cases (such as division by zero) -> Exceptions
* Semantic- produce wrong results

Values and types

* Value- piece of data in a program, belong to different types( int, float, complex…)
* To find type 🡪 type(“Hello”)
* You can’t add different types
* To convert just add in the beginning the acronym (f.e. : str(1+2)

Data types:

* Numeric
* Boolean
* Strings
* List
* Tuples
* Sets
* Dictionaries

(can define new data types- CLASSES)

Note: When you print values converts to String

Converting a float to int, truncates towards zero

For example:

>>>int(2.78)

2

Variables and Assignment

* Variable refers to a value
* Assignment statement assigns a value to a variable
* In an expression a variables is substituted by its value
* Some keywords can’t be used -> if given Syntax Error
* Reassignment🡪you can assign a value to an existing variable
* Uma imagem com mesa

  Descrição gerada automaticamenteOrder of execution is IMPORTANT

Operators, expressions and statements

* Operators( +, -, \*, \*\*…)
* Operands🡪 values combined by operators
* Operands must be compatible, result depends on their types
* Expression🡪 Combination of values, variables and operators that result in a value (a+=1)
* Statement🡪 Unit of Code that the Python interpreter can execute (print(a+=1). 🡪DOES NOT HAVE VALUE
* OPERATORS:
* \* - MULTIPLICATION
* \*\* - EXPONENTIATION
* / - DIVISION
* % - REST
* // - QUOCIENT

NOTE: + 🡪 CONCATENATION ON STRINGS

\* 🡪REPETITION ON STRINGS

# 🡪 COMMENTS

CALLING FUNCTIONS

* A function is a named sequence of statements that performs a computation
* Are invoked by name
* They take arguments ( in braces), that return a result or produce some effect

Uma imagem com texto

Descrição gerada automaticamente

Math Functions—use import math, and then when you want to use it, use math. followed by the function you want to use . ( f.e. : math.sin)

Input Function—gets input from console, ALWAYS string

Output Function—to ouput use print;

write muyltiple lines “\n”

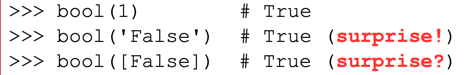
print multiple values (separated by blanks)

print(“speed=”, sep=” ”)

use sep= and end= to change how arguments are separated and terminated

Slide 2:

Boolean expression

* expression that is either True or False (special values🡪type bool)
* May be stored in variables, converted to strings or integer
* Null and empty values🡪 False
* Other🡪True

Uma imagem com texto

Descrição gerada automaticamenteRelational and logical operators

* Relational operators (==; !=; <;>) produce boolean results
* Three logical operators 🡪 and, or, not
* Precedence Rule🡪 **Arithmethic> Relational>Not>And>Or**

**Short Circuit Evaluation**—> operators AND and OR ONLY EVALUATE SECOND OPERAND IF NEEDED;

* in AND if first is False, second not evaluated;
* in OR if first is True, second not evaluated; therefore *order is IMPORTANT*

CONDITIONALS

* Allow the program check the conditions and change their behavior accordingly
* It may be of any type; preferred bool
* Can be nested🡪 better if deconstructed; see equivalences

IF: The indented statement is only executed if condition is TRUE; if FALSE doesn’t execute it

IF/ELSE: The indented statement is only executed if condition is TRUE; if FALSE doesn’t execute it; and executes the second indented statement.

IF/ELIF/ELSE: if more than two alternatives

Conditional Expression🡪

* based on a ternary operator

expression1 **if** condition **else** expression2

* is an expression
* Condition is evaluated first:
  + - if TRUE then expression2 is evaluated and is the RESULT;
    - if FALSE expression2 is evaluated and is the RESULT;

Uma imagem com texto

Descrição gerada automaticamente

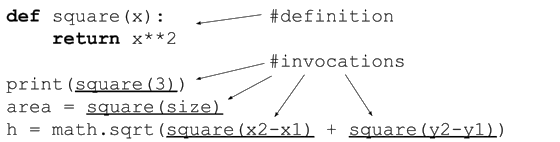
Slide 3:

Functions: we have to define them (header starts with def and ends with (:), indented body) and then we can call it several times if needed

Uma imagem com texto

Descrição gerada automaticamente

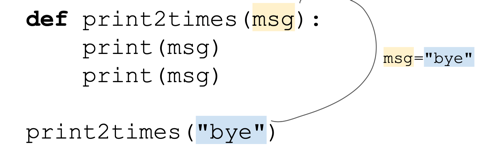
Function definition: specifies the name of a new function, list of parameters and block of statements to execute when called

**

Definition vs Invocation: definition statements are not executed, just stored for later use; in invocation is the only way they are executed.

*A function MUST be defined before being called.*

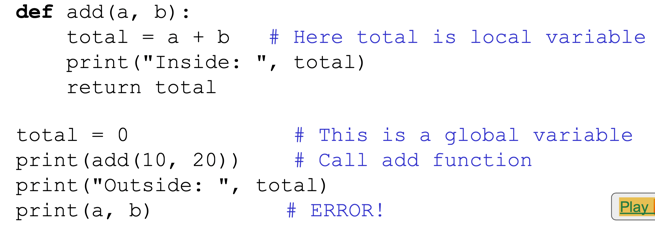
Flow of Execution: statements are executed from top to bottom, but where they find a function they skip it until that function is called, when it is executes the statements inside the function and then comes back from where it was left.

Parameters and arguments: some require arguments(what is inside ()).

When the function is called, the arguments are assigned to variables called parameters.

Return values- some functions produce results, others perform an action but don’t return values, like print.

The statement **return** *expression* exits from a function and returns the result of the expression, whereas a return with no arguments is a return none

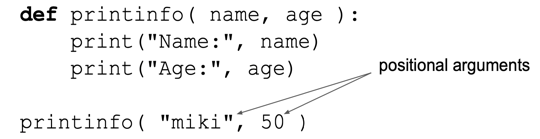
Global vs Local Variables

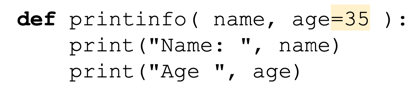
* Defined inside a function🡪local
* Defined outside a function🡪global
* Even when you have the same name inside a function and outside 🡪different variables

**Uma imagem com texto

Descrição gerada automaticamenteParameters is a local variable**: you can modify them but their effect is local. When function is called the parameter receives only the value of the argument.-->pass by value

Keywords Arguments:

* In a function call arguments are assigned to a parameter according to their position,
* If they have **keyword arguments** you just have to put the argument followed by = and then the info, *order is arbitrary* ( in this case **name=”miki”**)

When **definition specifies default argument values** for some of its parameters, when calling the function if a value it is not provided 🡪takes default value.

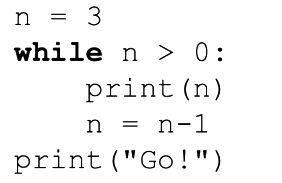
**Variable length arguments (advanced)-** when you need to process a function for more arguments than you’ve specified.

An asterisk(\*) is placed before the variable name that holds the values of all non-keyword variable arguments.

**Lambda**: is an expression whose result is a function

* you may store it in a variable;
* cannot contain statements only a single expression;
* most useful to pass as arguments to other functions

Slide 4

**Loops**

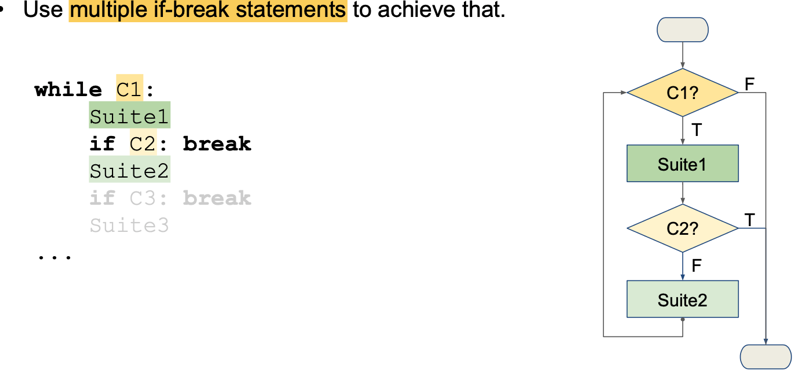
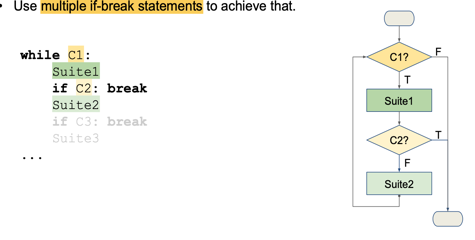
While- repeatedly executes the target statements until condition is False, when this happens the execution jumps out the loop.

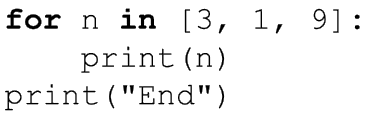
The *condition* (in while) should be a Boolean expression, else it is **implicitly converted** to bool, so null or empty means False.

**Note:** the body of the loop, should change conditions, in order that the *condition (above n>0) becomes False*, else it will repeat forever, which is called an infinite loop

Break-when you need the loop **to stop halfway**, and **jump out the loop**. Can only occur within for or while loop.(Loops with this pattern🡪loop and a half)

Continue🡪 **skips to the next iteration** of the enclosing loop body, *without executing the remaining statements in the current iteration.* Can only occur within for or while loop.

Multi-Exit loops(use of **multiple if-break statements** )🡪when there are *several conditions to terminate the loop and multiple places to test them along the body of it.*

For🡪 **repeats statement once for each item in a collection of items**, such as a list, a string or a tuple.

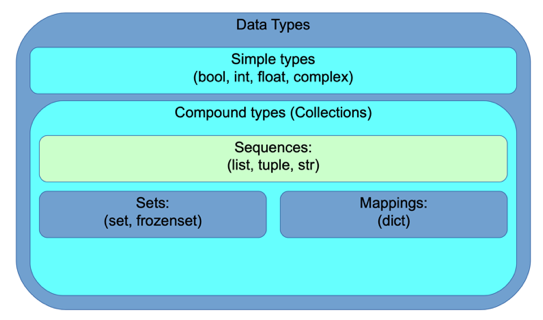
Uma imagem com texto

Descrição gerada automaticamente

**How is executed**: The collection is an expression and is *evaluated first*. Then, the first item in the collection is assigned to var, and the statements of block is executed once, the second item is assigned to var and executed again, and **so on until no more items** to assign to var in the collection.

Range🡪returns an object that generates a **sequence of integers in arithmetic progression. (range(10)🡪0 a 9)(range(10,0,-2)🡪 until 0 but without it )**

* Often used in for loops.
* May be called with *1, 2 or 3 arguments*
* All arguments must be **integers**
* Can be positive or negative
* Generated *up/down, but not stop.*



Slide 5:

**List**- mutable sequence of values (elements or items) of any type. Written in brackets ([]).

Examples: numbers=[1, 2, 3]

fruits=[“mango”, “banana”]

empty=[]

things=[“mango”, 1.5, [1,2]]. #list inside list

* len🡪indicates length
* Slicing🡪 can extract a subsequence. Example: numbers[1:3]. Negative indices may be used to. May be omitted at start (number[:2]🡪means from 0 to 1 (included)), or at end (number[2:]🡪means from 2 to end (included))
* Uma imagem com texto

  Descrição gerada automaticamenteTransverse🡪 use for loop, using indexes with for and range, or a while with the length of it. To transverse the items and indexes at the same time:

Sequence Operations:

* + concatenates (adds)
* repeats sequences.
* In checks if item is in list, contrary in not in
* Sum🡪 does the sum
* Min🡪finds the minimum
* Max 🡪finds maximum

Lists:

* Are mutable
* Can even change a list/add or take sublist
* Methods:

Uma imagem com mesa

Descrição gerada automaticamente

Mutability and Aliasing

* Uma imagem com texto

  Descrição gerada automaticamenteIn Python variables, store references to objects.
* If a is assigned to b, a=b 🡪**Aliases**- different names, refer to the same object. Therefore, if I modify a, it is changed under all names. If is changed inside the function, outside it is too🡪*Memory efficient*

Equality vs Identity

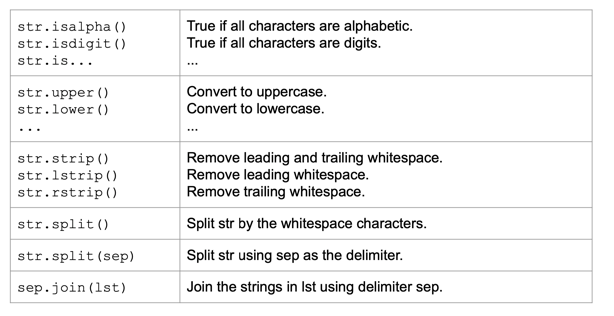
* Objects might be equal without being the same.
* Equality is tested with == whereas Identity is with in.
* Identity implies Equality. Equality does NOT imply Identity.

Note: Do not rely very much on **in**.

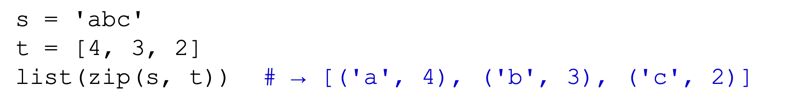
Cloning:

* Slicing Operator- [:]
* Copy Method- b=a.copy()
* Immutable types(Strings and tuples) don’t need one

Strings:

* Sequences of characters
* Delimited by “ ” or ‘ ’.
* Can use indexing and slicing
* Can concatenate and repeat
* But they are immutable, so you if you want to save you need to save in a new variable if not they will only return
* To transverse:
* While loop
* For loop
* Relational operators work on strings
* Characters are saved in numeric codes(according to *Unicode*) [ord(c)🡪 returns code of character]
* Methods:

**Tuples- t=(‘a’, ‘b, ‘c’)**

* Immutable sequence of values
* Values are indexed by integers(like in lists)
* To create a tuple with 1 element 🡪 t1=(“1”, ) [add comma in the end]
* To create an empty use the method **tuple(), what is inside is what is going to be in the tuple; if t=tuple(“banana”)** 🡪 **t=(“b”, “a”, “n”, “a”, “n”, “a”)**
* Zip--± combines tuples
* Relational Operators work 🡪 compares element to element
* Sorted/sort🡪 sort by first element if occurs a tie then second element and so on
* May be used as keys in dictionaries

Slide 6

**Text Files**

* Simplest way for program to maintain data is by reading and writing text files.
* Text File- sequence of characters stored in a persistent medium like hard drive, CD-ROM…
* Characters are encoded in bytes according to a certain coding table

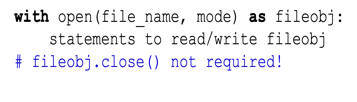
Opening and closing files

Must open the file before writing or reading

fileobj=open(file\_name, “r”) 🡪 r for reading/w for writing

More modes: “r”, “w”, “a”…

After using file, close it 🡪 fileobj.close()

 Using with, automatically closes

Text versus binary mode

* Normally files are opened in text mode
* Newline characters(“\n”) are converted to/from platform specific line endings
* Characters are encoded/decoded: each character is converted to/from one or more bytes
* May specify encoding🡪 fileobj=open(file\_name, “r”, encoding= “utf-8”)
* For files with no text use “wb” or “rb”🡪 you write/read strings of bytes (no conversions occur)

Uma imagem com texto

Descrição gerada automaticamenteReading a File

Use a for loop to read line by line:

Uma imagem com texto

Descrição gerada automaticamente

Use readline method:

 Read the entire file as string:

 Read at most N characters:

Moving the file objects’ position

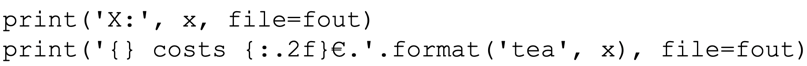
* tell() 🡪 tells you the current position within the file
* seek(offset) 🡪 changes the current position to offset bytes from the start.

Uma imagem com texto

Descrição gerada automaticamente

Write a file

* Use “w”🡪 creates a new file or deletes all old data and starts from scratch
* “a”🡪 does not delete, appends to the end of the file
* Write method (fout.write()) puts data into the file, if you can it again it will put the new data at the end, because the file object tracks the position
* Argument of write has to be string, convert it or format it
* May also use print with the file= argument



Filenames and paths

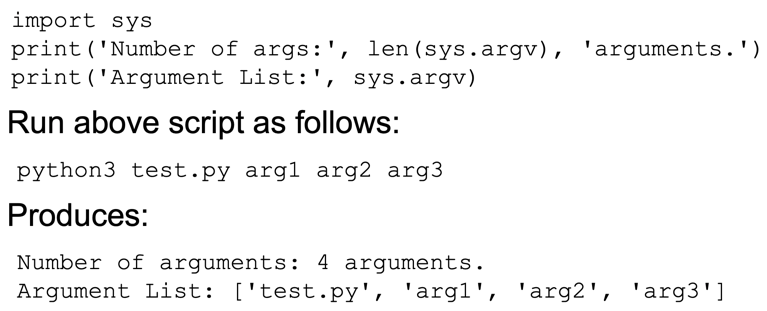
* The os modules🡪 provides functions for working with files and directories
* String that identifies a file🡪 path [os.getcwd() 🡪 returns name of current directory]
* Absolute path starts with /. 🡪 find it with os.path. abspath()
* Relative Path starts with the current directory

Functions with os 🡪 dar import

* os.path.exists(path) 🡪 checks if a file exists
* os.path.isdir(path)🡪 checks if filename is a directory
* os.path.isfile(path) 🡪 checks if regular file
* os.listdir()🡪 retruns a list of files in that directory

Command Line arguments

* The sys module provides acess to any command line arguments via the sys.argv variable
* len(sys.argv) 🡪 number of command line arguments
* sys.argv[0]🡪 program(script) name

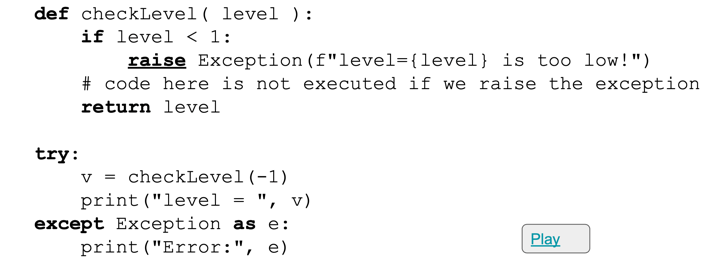


**Exceptions**

When Python encounters a situation it can’t deal with it raises an exception, interrupts the normal flow of the program

Information about the event is transmitted all the way through in an exception object

* Try: intercept select exceptions and resume normal execution
* An exception can have an argument which is a value that gives additional info about the problem
* Raise: used to raise exceptions



**Assertions**

* A condition that is required to be true
* It evaluates the condition if false raises an exception
* Can be placed in the beginning of the program to check for valid input

Slide 7

**Dictionaries**

* Unordered- order doesn’t matter, but stored in insertion order
* Associative – associates a key to a value
* Collection of unique (no two items can have the same key) items- may contain zero or more items
* Also called Associative Arrays or Maps
* Items also called key-value pairs
* Use {}
* To access value for a given key: shop[“sugar”] 🡪 # 1.0
* Mutable: shop[“sugar”]=6
* Values can be of any type but keys have to be immutable objects (not lists)🡪 Keys MUST BE HASHABLE

Dictionaries vs Lists

* Dictionary is kind of generalized list. In a list indices must be integers, in a dictionary any kind.
* Can’t take slices from dictionaries
* In lists order is IMPORTANT but in dictionaries not

Dictionaries Operations

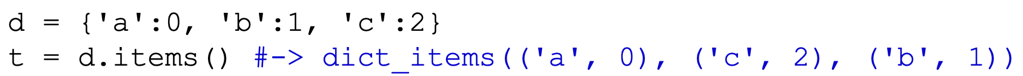
* len() 🡪 returns number of key-value pairs
* in 🡪 if something appears as key
* if want to see if something is a value in a dictionary 🡪 “uno” in eng2sp.values() 🡪 IT IS SLOW
* trying accessing inexistent key🡪 ERROR --± but using get will return default value
* setdefault is similar but creates a new item if missing
* pop(key) to remove item and return its value
* delete item with del 🡪 del d[20]
* popitem🡪 removes one unspecified item and returns the key-value pair

**Transverse Dictionary**

Uma imagem com texto

Descrição gerada automaticamenteFor:

Dictionaries and lists of tuples

* Method items returns a sequence of tuples, where each tuple is a key-value pair
* We can se a list of tuples to initialize a new dictionary

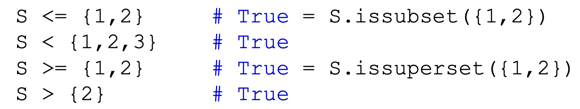
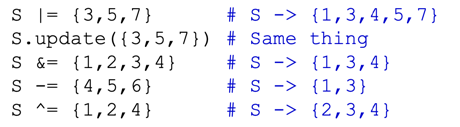
Slide 8:

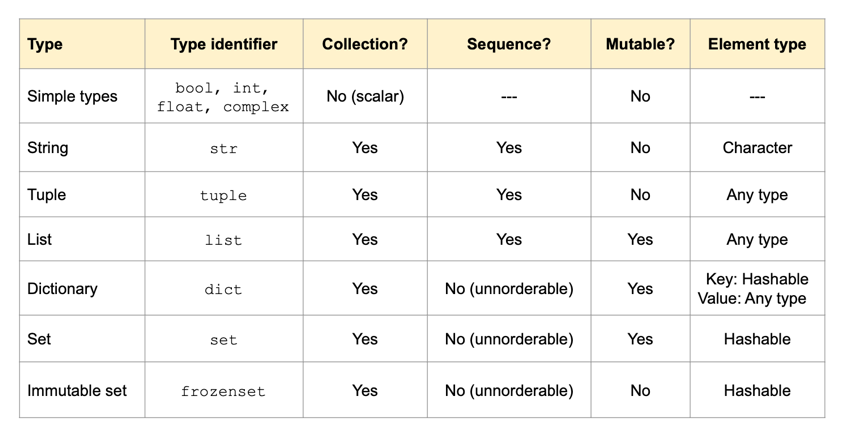
**Sets**

* ****Unordered (no order) collection(may contain zero or more elements) of unique items(cannot be repeated)
* The set constructor converts from other types
* Empty set MUST be created with set(). {} creates dictionary
* Common Application is for eliminating duplicates, but also eliminates order as a set doesn’t recall the entry of elements
* Indexing, Slicing and Concatenation NOT ALLOWED
* May contain elements of various types, but ONLY HASHABLE [lists, dictionaries or sets not allowed]
* Len()🡪 sees length
* Sets support intersection, union, set subtraction

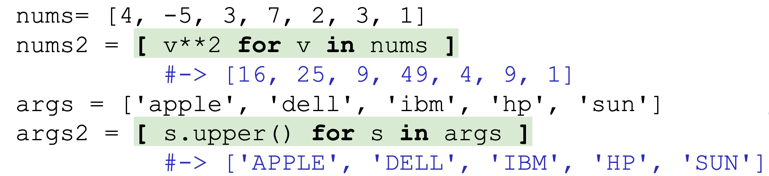
Uma imagem com texto

Descrição gerada automaticamente

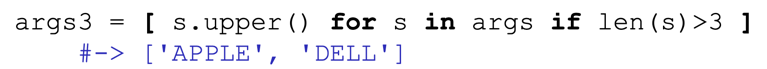
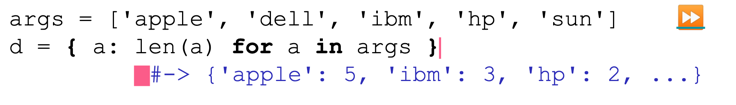
* May be compared (use ==)
* May test subset or superset relations
* ****Mutable: can add/remove items Or update it by union, intersection or difference:
* Or **frozenset()**🡪an **immutable set**

**GLOBAL:**

**Comprehensions**

* When you need to create collections with elements related to those in another

collection.

* Lists comprehensions🡪expressions that generate lists by operating on the elements of other collections
* May also include if clause
* Can create dictionary by comprehension

**Generator Expressions**

* Identical to list comprehensions but **enclosed in ()**
* They create an object that **generates items only if and when needed**

**Uma imagem com texto

Descrição gerada automaticamente**

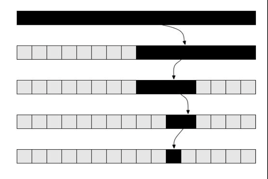
* May use these expressions to create other types of sequences



Slide 9

Searching Algorithm🡪 Binary Search🡪 MUCH FASTER

In a Sequential Search🡪 scans a sequence start to end or end to start

**Binary Search: L**

* Compares X to the element in the middle of L
* If X is smaller, searches only the first half of L
* If X is larger, searches only the second half of L

Using the bisect functions:

The bisect module includes functions that perform **binary search in a sorted list**

Functions to insert in order: bisect.insort

Sorting

* A sorted sequence is much faster to search
* Can be sorted through different criteria:
* Lexicographic
* Length
* Case-insensitive

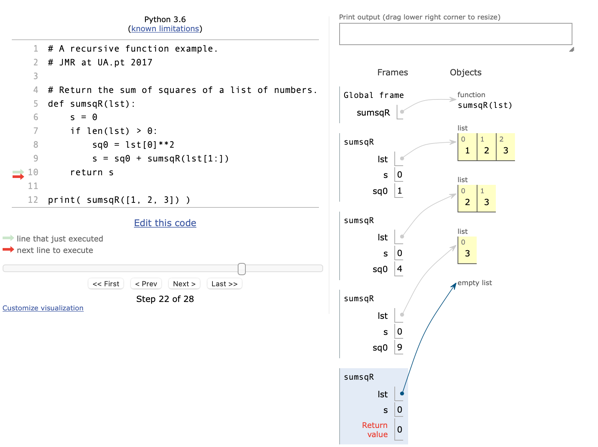
Optional key argument receives a function to sort elements by, to reverse use reverse=True. Note: tuples can be sorted too--± compared like strings form left to right

**Lambda expressions**

* Define simple anonymous functions
* Result must be an expression.
* Should only be used for simple expressions
* Useful to pass as arguments (key=..)

Insertion Sort:

1. Assume the first K elements are sorted
2. Save L[k] in T
3. Insert T, in order, into first K elements (overwriting L[K])
   1. Move every L[J]>T to L[J+1], starting from J=K-1
   2. Put T into the vacant slot
4. Now, increment K and repeat



Slide 10:

**Recursive Functions:** a function that calls itself

* Notice that the function has 4 variables lst, 4 s, 3sq0, but all distinct.
* Each frame stores the local context of a single function call
* Frames are stored in the program stack

**Rules:**

* Must have **Base Cases**: cases that can be resolved without recursive calls
* Context passed to the recursive call **must be different** than context received
* In successive recursive calls, **context must converge towards the base cases**

**Note:** Recursive usually requires extra time and memory cost(function call and memory usage)

**Guidelines to write a recursive function:**

1. Define arguments: what you need, what they mean, result you expect
2. Assume the function will work: describe how a solution to a problem can be obtained by modifying the solutions to smaller versions of the program(recursive part)
3. Determine base cases: which have trivial answer