# Python variables

|  |  |
| --- | --- |
| **Example** | **Type** |
| Last\_Name = Delgado | String |
| Score = 9.83 | Float |
| Approved = True | Boolean |
| Number\_List = 3 | Int |

# Strings

|  |  |
| --- | --- |
| **Example** | **Description** |
| Course\_Name = “Ultimate python” | We use “ ” to print character strings. |
| Course\_Description = “ “ “ This course teaches python skills to get a job as a programmer ” ” ” | We use “ “ “ ” ” ” to print to many character strings. |
| Course\_Name = “Ultimate python”  print(len(Course\_Name)) | We use len(Course\_Name) to print the length of the character string we assign to the in a variable.  print(len(Course\_Name)) 🡪 15  Length is 15 characters |
| Course\_Name = “Ultimate python”  print(Course\_Name[0]) | We use [0], [1], [2], [3], …[n]. inside print to print the specific character of the character string.  print(Course\_Name[0]) 🡪 U  print(Course\_Name[1]) 🡪 I  print(Course\_Name[2]) 🡪 T  print(Course\_Name[9]) 🡪 P |
| Course\_Name = “Ultimate python”  print(Course\_Name[0:8]) | We use [n:n] inside print to cut and print the characters strings.  print(Course\_Name[0:8]) 🡪 Ultimate  print(Course\_Name[9:]) 🡪 Python  print(Course\_Name[:8]) 🡪 Ultimate  print(Course\_Name[:])🡪 Ultimate Python |
| Name = Ozzy  Last\_Name = Aceves  Full\_Name = f ”{ Name } { Last\_Name } ” | When we want to format strings, we use  Full\_Name = f ”{ Name } { Last\_Name } to format strings.  When we have:  Full\_Name = f ”{ Name } { Last\_Name }  print(Full\_Name) 🡪 Ozzy Aceves  When we have:  Full\_Name = f ”{ Name[0] } { 2 + 5 }  print(Full\_Name) 🡪 O 7 |
| Animal = “Happy Monkey”  print(Animal.upper()) | We use .upper to convert all characters to uppercase within the string.  print(Animal.upper()) 🡪 HAPPY MONKEY |
| Animal = “Happy Monkey”  print(Animal.lower()) | We use .lower to convert all characters to lowercase within the string.  print(Animal.lower()) 🡪 happy monkey |
| Animal = “Happy Monkey”  print(Animal.capitalize()) | We use .capitalize to convert the first character in uppercase within the string.  print(Animal.capitalize()) 🡪 Happy monkey |
| Animal = “Happy Monkey”  print(Animal.title()) | We use .title to convert the firsts characters in uppercase within the string.  print(Animal.title()) 🡪 Happy Monkey |
| Animal = “Happy Monkey”  print(Animal.strip()) | We use .strip to remove spaces at the beginning and at the end of the string.  print(Animal.strip()) 🡪 Happy Monkey |
| Animal = “Happy Monkey”  print(Animal.capitalize().strip()) | We use .capitalize().strip() (or other combination) to convert the first character in uppercase within the string and remove spaces at the beginning and at the end of the string.  print(Animal.capitalize().strip()) 🡪 Happy monkey |
| Animal = “ Happy Monkey”  print(Animal.lstrip()) | We use .lstrip to remove the spaces to the left of the string.  print(Animal.lstrip()) 🡪 Happy Monkey |
| Animal = “Happy Monkey”  print(Animal.rstrip()) | We use .rstrip to remove the spaces to the right of the string.  print(Animal.rstrip())🡪 Happy Monkey |
| Animal = “Happy Monkey”  print(Animal.find(“pp”)) | We use .find(“pp”) to search characters within the strings and return the indice.  print(Animal.find(“pp”)) 🡪 3  print(Animal.find(“zy”)) 🡪 -1 this case not found the characters and return -1 |
| Animal = “Happy Monkey”  print(Animal.replace(“pp”, “k”)) | We use .replace(“pp”, “k”) to replace characters within the strings.  print(Animal.replace(“pp”, “k”)) 🡪 Haki Monkey |
| Animal = “Happy Monkey”  print( “pp” in Animal) | We use print( “pp” in Animal) to search characters within the string and return a boolean (True or False).  print( “pp” in Animal) 🡪 True  print( “zy” in Animal) 🡪 False |
| Animal = “Happy Monkey”  print( “pp” not in Animal) | We use print( “pp” not in Animal) to search characters within the string and return a boolean (True or False).  print( “pp” not in Animal) 🡪 False  print( “zy” not in Animal) 🡪 True |

# Sequences and escapes

|  |  |
| --- | --- |
| Course = “Ultimate \”Python”\”  print(Course) | We use \“ \” when we want to write the character “ ” within strings and another “ ”.  print(Course) 🡪 Ultimate “Python” |
| Course = ’Ultimate ”Python”’  print(Course) | We use ‘ ’ when we want to write the character “ ” within strings and another “ ”.  print(Course) 🡪 Ultimate “Python” |
| #Comment | We use the symbol # to comment the code |
| Course = “Ultimate \nPython”  print(Course) | We use \n when we want to write the following characters in the following line.  print(Course) 🡪 Ultimate  Python |
| Course = “Ultimate \\Python”  print(Course) | We use \\ when we want to write the character \ within strings.  print(Course) 🡪 Ultimate \Python |

# Numbers

|  |  |
| --- | --- |
| Number\_1 = input( “Enter the number:” ) | We use input() when we want to store a data in a variable |
| Number\_Integer = 2 | This number is integer |
| Number\_Float = 4.2 | This number is float |
| Number\_Imaginary = 2 + 2j | This number is a imaginary |
| Number\_Integer = 2  Number\_Integer += 5 | The expression Number\_Integer = Number\_Integer + 5 can be written in another form.  Number\_Integer += 5 |
| Number\_Integer\_2 = 4  Number\_Integer\_2 -= 5 | The expression Number\_Integer = Number\_Integer - 5 can be written in another form.  Number\_Integer -= -1 |
| Number\_Integer\_3 = 6  Number\_Integer\_3 \*= 5 | The expression Number\_Integer = Number\_Integer \* 5 can be written in another form.  Number\_Integer \*= 30 |
| Number\_Integer\_4 = 8  Number\_Integer\_4 += 5 | The expression Number\_Integer = Number\_Integer / 5 can be written in another form.  Number\_Integer /= 1.6 |
| print(1 // 3) | We use // in the operation to print the result without decimals.  print(1 // 3) 🡪 0 |
| print(1 % 3) | We use % in the operation to print the residue of the division.  print(1 % 3) 🡪 1 |
| print(1 \*\* 3) | We use \*\* in the operation as an operator raised to power.  print(1 \*\* 3) 🡪 1 |
| print(round(1.2))  print(round(1.5))  print(round(1.7)) | We use round(n) in the operation to round off the result.  print(round(1.2)) 🡪 1  print(round(1.5)) 🡪 2  print(round(1.7))🡪 2 |
| print(abs(-77))  print(abs(55)) | We use abs(n) in the operation to print the absolute value.  print(abs(-77)) 🡪 77  print(abs(55)) 🡪 55 |
| Import math  print(math.ceil(1.1)) | We use math.ceil(n) in the operation to round up the number.  print(math.ceil(1.1)) 🡪 2 |
| Import math  print(math.floor (1.999)) | We use math.floor(n) in the operation to round down the number.  print(math.floor(1.999)) 🡪 1 |
| Import math  print(math.isnan(23)) | We use math.isnan(n) to print whether or not the value is a number.  print(math.isnan(23)) 🡪 False |
| Import math  print(math.pow(10, 3)) | We use math.pow(n) in the operation to raise a number to a certain power.  print(math.pow(10, 3)) 🡪 1000.0 |
| Import math  print(math.sqrt(9)) | We use math.sqrt(n) in the operation to get the square root of the number.  print(math.sqrt(9)) 🡪 3 |

# Type conversion

|  |  |
| --- | --- |
| Number\_1 = input( “Enter the number:” )  Number\_1 = int(Number\_1)  print(int(Number\_1)) | We use Number\_1 = int(Number\_1) to convert a string into an integer.  We use print(int(Number\_1)) to convert and print a string into an integer. |
| Number\_1 = input( “” )  Number\_1 = float(Number\_1)  print(float(Number\_1)) | We use Number\_1 = float(Number\_1) to convert a string into a float.  We use print(float(Number\_1)) to convert and print a string into a float. |
| Number\_2 = input()  Number\_2 = str(Number\_2)  print(str(Number\_2)) | We use Number\_2 = str(Number\_2) convert a data into a string.  We use print(str(Number\_2)) to convert and print a data into a string. |
| print(bool(“”))  print(bool(“ “))  print(bool(“0”))  print(bool(None))  print(bool(0)) | We use print(bool()) to convert and print a data into bool.  print(bool(“”)) 🡪 False  print(bool(“ “)) 🡪 True  print(bool(“0”)) 🡪 True  print(bool(None)) 🡪 False  print(bool(0)) 🡪 False |

# Logic comparators

|  |  |
| --- | --- |
| print(1 < 2) | We use < to indicate that the data or value is less than another one.  print(1 < 2) 🡪 True  print(3 < 2) 🡪 False |
| print(1 > 2) | We use > to indicate that the data or value is greater than another one.  print(1 > 2) 🡪 False  print(3 > 2) 🡪 True |
| print(1 <= 2) | We use <= to indicate that the data or value is less than or equal to another.  print(1 <= 2) 🡪 True  print(2 <= 2) 🡪 True  print(3 <= 2) 🡪 False |
| print(1 >= 2) | We use >= to indicate that the data or value is greater than or equal to another.  print(1 >= 2) 🡪 False  print(2 >= 2) 🡪 True  print(3 >= 2) 🡪 True |
| print(1 == 2) | We use == to indicate that the data or value is equal to another one.  print(1 == 2) 🡪 False  print(2 == 2) 🡪 True  print(2 == “2”) 🡪 False |
| print(1 != 2) | We use != to indicate that the data or value is different from another one.  print(1 != 2) 🡪 True  print(2 != 2) 🡪 False  print(2 != “2”) 🡪 True |

# If, elif, else

|  |  |
| --- | --- |
| 1. Age = 33 2. if Age > 53: 3. print(“You can watch the movie at a discount”) 4. elif Age > 21: 5. print(“You can watch the movie”) 6. else: 7. print(“You can’t enter”) 8. print(“You must go elsewhere”) 9. print(“End”) | We use if to evaluate a statement, and elif to add another statement and so on and so forth, and we add else in case none of the above sentences are met. |
| 1. Age = 17 2. Message = "You are of legal age" if Age > 17 else "You are not of legal age" 3. print(Message) | We use Message = "You are of legal age" if Age > 17 else "You are not of legal age" when using ternary operator we assign values depending on the condition to a variable. |

# Logical Operators

|  |  |
| --- | --- |
| and | We use and to make a condition where we evaluate two conditions and if these two are fulfilled the main condition is fulfilled. |
| or | We use or to make a condition where we evaluate two conditions and if one or the other is met the main condition is met. |
| not | We use not to negate the boolean value of a variable. |

# For, For else, While

|  |  |
| --- | --- |
| 1. for number in range(5):   2. print(number) | We use for to make a loop until the condition is met.  We use range(5) to iterate the numbers from 0 to 5.   1. for number in range(5): 2. print(number)   1  2  3  4  5 |
| 1. for char in “Hello”:   2. print(char) | here is an example how to print character for character of the strings.  H  e  l  l  o |
| 1. search = 3 2. for number in range(5): 3. print(number) 4. if number == search: 5. print(“found”, search) 6. break 7. else: 8. print(“Not found”) | We use if inside for to make a loop and look for data, then we use break to stop the loop and not continue with the loop, in case of not finding the data, we use else.  1  2  3  Found 3 |
| 1. while number < 100: 2. print(number) 3. Number \*= 2 | We use while to loop if a condition is met.  1  2  4  8  16  32  64 |
| 1. while True: 2. command = input(“$ ”) 3. print(command) 4. if command.lower() == “exit” 5. break | We can use while with a boolean value, but we must add a conditional break so as not to crashe the program. |
| 1. for j in range(3): 2. for k in range(2): 3. print(f ”{ Name } { Last\_Name } ”) | We use for inside another for to nest two loops. |

# Functions

|  |  |
| --- | --- |
| 1. def Hello(): 2. print(“Hello world”) 3. print(“Ultimate Python”) 4. Hello() | We use def to define a functions, inside the functions there are processes that are executed when they are called throughout the code. |
| 1. def Hello(Name, Last\_Name=“Mclovin”): 2. print(“Hello world”) 3. print(f “Welcome { Name } { Last\_Name }”) 4. Hello(“Ozzy” , “Aceves”) 5. Hello(“Petu”) | We can use variables def Hello(Name, Last\_Name) when we declare the name of the functions and arguments Hello(“Ozzy” , “Aceves”) when we send to call the functions.  Hello(“Ozzy” , “Aceves”) 🡪 Hello world Welcome Ozzy Aceves  Hello(“Petu”) 🡪 Hello world  Petu Mclovin |
| 1. def Hello(Name, Last\_Name=“Mclovin”): 2. print(“Hello world”) 3. print(f “Welcome { Name } { Last\_Name }”) 4. Hello(“Ozzy” , “Aceves”) 5. Hello(“Petu”) 6. Hello(Name=“Aceves” , Last\_Name=“Ozzy”) | We can use named arguments Hello(Name=“Aceves” , Last\_Name=“Ozzy”) when we send to call functions to indicate the order of the variables.  Hello(Name=“Aceves” , Last\_Name=“Ozzy”) 🡪 Hello world  Welcome Ozzy Aceves |
| 1. def addition(\*numbers):   2. result = 0  3. for number in numbers:  4. result += number  5.  6. addition(2, 3, 4,)  7. addition(22, 12, 14, 0, 5)  8. addition(4, 5) | We use \* in the para parameters when defining a function, this makes the parameters iterable and multiple values can be assigned when the function is called  6. addition(2, 3, 4,) 🡪 9  7. addition(22, 12, 14, 0, 5) 🡪 53  8. addition(4, 5) 🡪 9 |
| 1. def get\_product(\*\*data): 2. print(data) 3. def get\_product\_2(\*\*data): 4. print(data[“id”], data[“name”]) 5. get\_product(id=“id” , name=“iPhone”, desc=“smartphone”) 6. get\_product\_2(id=“1341” , name=“Samsung”, desc=“smartphone”) | We use \*\* in the parameters when defining a function, this makes the parameters iterable and multiple values can be assigned when calling the function, otherwise, packing multiple parameters into one.  7. get\_product(id=“id” , name=“iPhone”, desc=“smartphone”) 🡪 ‘id’ : ‘id’, ‘name’ : ‘iphone’, ‘description’ : ‘This is an iPhone’  8. get\_product\_2(id=“1341” , name=“Samsung”, desc=“smartphone”) 🡪 1341 Samsung |
| 1. def addition(a, b): 2. result = a + b 3. return result 4. c = addition(1, 2) 5. d = addition(c, 2) 6. print(d) | we use the return reserved word to be able to return the value of a variable, this allows us to use the returned value of a variable of a certain function in some other function.  5. c = addition(1, 2) 🡪 c = 3  6. d = addition(c, 2) 🡪 d = 5  7. print(d) 🡪 5 |
| 1. greet1 = “Hello Dogo” 2. def greet(): 3. global greet1 4. greet1 = “Hello World” 5. print(greet1) 6. def greetpetu(): 7. greet1 = 24 8. print(greet1) 9. print(greet1) 10. greet() 11. print(greet1) | We use global <variable\_name> inside functions to use variables declared outside functions, however, these can be overwritten if they interact in a function, which may cause errors.  11. print(greet1) 🡪 Hello Dogo  12. greet() 🡪 Hello World  13. print(greet1) 🡪 Hello World |

# Lists

|  |  |
| --- | --- |
| 1. Numbers = [2, 5, 7, 10, 1, 6, 14] 2. Letters = [“a”, “d”, “H”, “p”] 3. Words = [“Hello”, “Cat”, “Miss”] 4. Booleans = [True, False, True, False] 5. Matrix = [[1, 5], [3, 9]] 6. Zero = [0] \* 10 7. Alphanumeric = Numbers + Letters 8. Range = list(range(1, 11)) 9. Chars = list(“Hello Dogo”) 10. print(Numbers) 11. print(Letters) 12. print(Words) 13. print(Booleans) 14. print(Matrix) 15. print(Zero) 16. print(Alphanumeric) 17. print(Range) 18. print(Chars) | We use lists to store numbers, characters, strings. These can be combined and written in different ways  10. print(Numbers) 🡪 [2, 5, 7, 10, 1, 6, 14]   1. print(Letters) 🡪 [“a”, “d”, “H”, “p”] 2. print(Words) 🡪 [“Hello”, “Cat”, “Miss”] 3. print(Booleans) 🡪 [True, False, True, False] 4. print(Matrix) 🡪 [[1, 5], [3, 9]] 5. print(Zero) 🡪 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] 6. print(Alphanumeric) 🡪 [2, 5, 7, 10, 1, 6, 14, 'a', 'd', 'H', 'p'] 7. print(Range) 🡪 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 8. print(Chars) 🡪 ['H', 'e', 'l', 'l', 'o', ' ', 'P', 'e', 't', 'u'] |
| 1. Pet = [“Petu”, “Merlyn”, “Bibi”, “Dogo”] 2. print(Pet) 3. print(Pet[2]) 4. Pet[2] = “Fausto” 5. print(Pet) 6. print(Pet[2:]) 7. print(Pet[1:2]) 8. print(Pet[-1]) 9. print(Pet[::2]) 10. Numbers = list(range(21)) 11. print(Pet[1::2]) 12. print(Pet[::2]) | We can modify the lists and print certain values from them, using the following syntax in each of them   1. print(Pet) 🡪 ['Petu', 'Merlyn', 'Bibi', 'Dogo'] 2. print(Pet[2]) 🡪 Bibi 3. Pet[2] = “Fausto” 🡪 Replace "Bibi" with "Fausto" 4. print(Pet) 🡪 ['Petu', 'Merlyn', 'Fausto', 'Dogo'] 5. print(Pet[2:]) 🡪 ['Fausto', 'Dogo'] 6. print(Pet[1:2]) 🡪 ['Merlyn'] 7. print(Pet[-1]) 🡪 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19] 8. print(Pet[::2]) 🡪 [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]   12.print(Pet[1::2]) 🡪 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]  13.print(Pet[::2]) 🡪 [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20] |
| 1. Numbers = [1, 2, 3] 2. First, Second, Third = Numbers 3. print(First, Second, Third) 4. Only\_First, \*Others = Numbers 5. print(Only\_First) 6. print(Others) 7. Numbers\_2 = list(range(1, 11)) 8. First, \*Others, Tenth = Numbers\_2 9. print(First, \*Others, Tenth) | When a list continues several numbers and we want to access each of them, we can use a variable which takes a certain value from the list and another that can store the rest.  3. print(First, Second, Third) 🡪 1, 2, 3  5. print(Only\_First) 🡪 1  6. print(Others) 🡪 [2, 3]  9. print(First, \*Others, Tenth) 🡪 1, [2, 3, 4, 5, 6, 7, 8, 9] 10 |
| 1. Pet = [“Petu”, “Merlyn”, “Fausto”, “Tazmania”] 2. Pet\_2 = [“Pimienta”, “Dogo”, “Kenich”, “Ceniza”] 3. for Pet in enumerate(Pet): 4. print(Pet) 5. for index, Pet\_2 in enumerate(Pet\_2): 6. print(index, Pet\_2) | We can use for to iterate lists and the enumerate function to be able to iterate each of the items in the lists, but the enumerate function will give two values, the index, represented by a number depending on the order of the item and the item.  For lines 3 and 4:  🡪 (0, 'Petu')  🡪 (1, 'Merlyn')  🡪 (2, 'Fausto')  🡪 (3, 'Tazmania')  For lines 4 and 6, if we didn't print the index variable it would look without the index of the enumerate function (without a number)  🡪 0 Pimienta  🡪 1 Dogo  🡪 2 Kenich  🡪 3 Ceniza  Without a number  6. print(Pet\_2)  🡪 Pimienta  🡪 Dogo  🡪 Kenich  🡪 Ceniza |
| 1. Pet = [“Petu”, “Merlyn”, “Dogo”, “Fausto”, “Tazmania”] 2. print(Pet.count(“Dogo”)) 3. if “Dogo” in Pet: 4. print(Pet.index(“Dogo”)) | We can look up some value in the lists if we use the if condition and look for where it is with index.  4. 🡪 2 |
| 1. Pet = [“Petu”, “Merlyn”, “Dogo”, “Fausto”, “Tazmania”] 2. Pet.inser(3, “Manji”) 3. Pet.append(“Manji”) 4. Pet.remove(“Dogo”) 5. Pet.pop(2) 6. del Pet[3] 7. Pet.clear() | We can add or remove items from the list, using the following types.   1. Pet.inser(3, “Manji”) 🡪 Add the value "Manji" to position 3 in the list. 2. Pet.append(“Manji”) 🡪 Add the value "Manji" to the end of the list. 3. Pet.remove(“Dogo”) 🡪 Remove the "Dogo" value from the list 4. Pet.pop(2) 🡪 Delete item 2 in the list, if it doesn't have a number, delete the last one. 5. del Pet[3] 🡪 Removes the item from position 3 in the list. 6. Pet.clear() 🡪 Remove all items from the list |
| 1. Numbers = [2, 15, 42, 4] 2. Users = [[4, “Ozzy”], [15, “César”], [10, “Paolo”]] 3. Users\_2 = [[“Ozzy”,4], [“César”, 15], [“Paolo”, 10]] 4. Numbers.sort() 5. Numbers.sort(reverse=True) 6. Numbers\_2 = sorted(Numbers) 7. Numbers\_2.sorted(Numbers, reverse=True) 8. Users.sort() 9. def Sorting(item): 10. return item[1] 11. Users\_2.sort(key= Sorting) | We can sort the items in the lists, as long as they have a sorting element, such as numbers  4. Numbers.sort() 🡪 [2, 4, 15, 42]  5. Numbers.sort(reverse=True) 🡪 [42, 15, 4, 2]  6. Numbers\_2 = sorted(Numbers) 🡪 [2, 4, 15, 42] Sorts data but another variable is created, preserving the original list.  7. Numbers\_2 = sorted(Numbers, reverse=True) 🡪 [42, 15, 4, 2]  8. Users.sort() 🡪 [[4, 'Ozzy'], [10, 'Paolo'], [15, 'César']] We can sort lists that contain lists inside, as long as they have as their first element a value that can be sorted, such as a number.  11. Users\_2.sort(key= Sorting) 🡪 [['Ozzy', 4], ['Paolo', 10], ['César', 15]] |
| Users\_2.sort(key=lambda el: el[1]) | We can use lambda functions when you need to execute a function only once in the entire code, this function must be given two arguments, the first is the parameters it receives, and then the content of the merge, in this example the value of the return.  Users\_2.sort(key=lambda el: el[1]) = def Sorting(item):  return item[1]  Users\_2.sort(key=lambda el: el[1]) 🡪 [['Ozzy', 4], ['Paolo', 10], ['César', 15]] |
| 1. Users = [[4, “Ozzy”], [15, “César”], [10, “Paolo”]] 2. Users\_2 = [[“Ozzy”,4], [“César”, 15], [“Paolo”, 10]] 3. Users\_3 = [[“Hacksiel”, 8], [“Roberto”, 9], [“Minutti”, 6]] 4. Names = [] 5. for User in Users: 6. Names.append(User[0]) 7. print(Names) 8. Names\_2 = [Users\_2[0] for User\_2 in Users\_2] 9. print(Names\_2) 10. Names\_3 = [Users\_3[0] for User\_3 in Users\_3 if Users\_3[1] > 7] 11. print(Names\_3) | We can transform lists, filter lists, and combine both actions by using the for loop with conditionals.  7. print(Names) 🡪 ['Ozzy', 'César', 'Paolo']  8. Names\_2 = [Users\_2[0] for User\_2 in Users\_2] 🡪 ['Ozzy', 'César', 'Paolo']  10. Names\_3 = [Users\_3[0] for User\_3 in Users\_3 if Users\_3[1] > 7] 🡪 ['Petu', 'Tazmania', 'Merlyn'] |
| 1. USER = [[“0014”, 8], [“0004”, 9], [“1341”, 6]] 2. USER\_2 = [[“0014”, 8], [“0004”, 9], [“1341”, 6]] 3. NAMES = list(map(lambda USER: USER[0] , USERS)) 4. print(NAMES) 5. Less\_Users = list(filter(lambda USER\_2: USER\_2[1] >7, USERS\_2)) 6. print(Less\_Users) | We use map and filter functions to transform and filter lists.  With map we transform the list we need  3. NAMES = list(map(lambda USER: USER[0] , USERS)) 🡪 ['0014', '0004', '1341']  With filter we filter the lists we need  5. Less\_Users = list(filter(lambda USER\_2: USER\_2[1] >7, USERS\_2)) 🡪 [[“0014”, 8], [“0004”, 9] |
| 1. Numbers = (1, 2, 3) + (4, 5, 6) 2. print(Numbers) 3. Not\_modify = tuple([1, 11, 22, 33]) 4. print(Not\_modify) 5. less\_numbers = Numbers[:2] 6. print(less\_numbers) 7. for i in Numbers: 8. print(i) 9. List\_numbers = list(Numbers) 10. List\_numbers[0] = “Petu” 11. print(list\_numbers) | Tuples are like lists, however the data they contain cannot be modified, to use tuples they are declared with parentheses () instead of square brackets [], or we use the word tuple.  2. print(Numbers) 🡪 (1, 2, 3, 3, 5, 6)  4. print(Not\_modify) 🡪 (1, 11, 22, 33)  6. print(less\_numbers) 🡪 (1, 2)  8. print(i) 🡪 1 2 3 4 5 6  11. print(list\_numbers) 🡪 ['Petu', 2, 3, 3, 5, 6] |
| 1. first = {1, 1, 2, 2, 3, 4} 2. print(first) 3. second = [3, 4, 5] 4. second = set(second) 5. print(second) 6. print(first | second) 7. print(first & second) 8. print(first - second) 9. print(first ^ second) | Sets, like lists and tuples, contain data, unlike lists and tuples, which remove the repeated data they contain, they are declared with curly brackets {}. The sets can be combined with each other, using different symbols.  2. print(first) 🡪 {1, 2, 3, 4} Prints the data inside without repeating.  5. print(second) 🡪 {3, 4, 5} Prints data that was previously a list.  6. print(first | second) 🡪 {1, 2, 3, 4, 5} Combine data from both sets  7. print(first & second) 🡪 {3, 4} Print the data in common between both sets  8. print(first - second) 🡪 {1, 2} Removes common data from the first set  9. print(first ^ second) 🡪 {1, 2, 5} Combines data that is not common across sets |
| 1. Point = {“X”: 15, “Y”: 20} 2. print(Point) 3. print(Point[“X”]) 4. print(Point[“Y”]) 5. Point[“Z”] = 45 6. print(Point) 7. if “W” in Point: 8. print(“I’ve found W: ”, Point) 9. print(Point.get[“X”]) 10. print(Point.get[“W”], “Doesn’ t exist”) 11. del Point[“X”] 12. print(Point) 13. Point[“X”] = 15 14. for value in Point: 15. print(value, Point[value]) 16. for value in Point.items(): 17. print(value) 18. for key, value in Point.items(): 19. print(key, value) 20. Users = [ 21. {“ID”: 1, “Name”: “Petu”} 22. {“ID”: 2, “Name”: “Merlyn”} 23. {“ID”: 3, “Name”: “Dogo”} 24. {“ID”: 4, “Name”: “Fausto”} 25. ] 26. for user in Users: 27. print(user[“Name”]) | Dictionaries are lists that are represented by two values, in this case the first argument (the key) is a string, followed by the value which can be anything. Point = {“X”: 15, “Y”: 20}, y utilizamos x para conver  2. print(Point) 🡪 {'X': 15, 'Y': 20}  3. print(Point[“X”] 🡪 15  4. print(Point[“Y”]) 🡪 20  6. print(Point) 🡪 {'X': 15, 'Y': 20, 'Z': 45}  8. print(“I’ve found W: ”, Point) 🡪  9. print(Point.get[“X”]) 🡪 15  10. print(Point.get[“W”], “Doesn’ t exist”) 🡪 Doesn’t exist  12. print(Point) 🡪 {'Y': 20, 'Z': 45}  15. print(value, Point[value]) 🡪 Y 20 Z 45 X 15  17. print(value) 🡪 (‘Y’, 20) (‘Z’, 45) (‘X’, 15)  19. print(key, value) 🡪 Y 20 Z 45 X 15  27. print(user[“Name”]) 🡪 Petu Merlyn Dogo Fausto |
|  |  |