# 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”  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) | Cuando una lista continue varios números y queremos acceder a cada uno de ellos podemos utilizar una variable la cual tome cierto valor de la lista y otra que pueda almacenar el resto.  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 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |