7-1-2025 - 10-1-2025

ALO

**\_Documentacion mi codigo para aprender y practicar PYTHON**

Nota – saque la lista de archivos .PY usando CMD.EXE y comando: “dir \*.py /od >"lista\_archivos\_de\_python.txt"

**07/11/2024 08:04 a.m. 102**

**Programa 1)**

**input.py**

print("Welcome to the greeter program")

name = input("Enter your name: ")

print("Greetings " + name)

- El comando print con un parámetro entre comillas imprime un string

- el comando input toma un string de la consola del usuario

- con print y varios parámetros unidos con + concatena datos del mismo tipo. En este caso, el string Greetings se une con la variable name que también tiene un string.

**07/11/2024 08:04 a.m. 45**

**Programa 2)**

**hello.py**

print("Hello World!")

print('Hello, World!')

- Los comandos anteriores muestran que se puede imprimir strings con comando print usando strings con comillas dobles al igual que con comillas simples con el mismo resultado. Inclusive con triples comillas dobles creo que se puede obtener el mismo resultado para usar strings

Que se quieran imprimir o usar con contenido que tenga comillas dobles o simples y al usar triples comillas no hay confusiones con strings que tengan comillas simples o dobles sin necesidad de usar caracteres de escape como \

**07/11/2024 08:04 a.m. 121**

**Programa 3)**

**datetest.py**

from datetime import date

print(date.today())

print(type(date.today()))

print("Today's date is: " + str(date.today()))

- En este programa uso el modulo datetime y se importa función date para trabajar con fechas.

- La segunda línea usa el método today de la función date importada para imprimir la fecha de hoy.

- La tercera línea nos dice con type que la clase de date.today() es datetime.date

- La cuarto línea une el string entre dobles comillas con la conversión a string usando str del objeto date.today() porque el tener formato de fecha requiere conversión a string para que el comando print no de error. El comando print puede unir o concatenar o procesar con + sólo datos del mismo tipo. Así por ejemplo, el print puede unir string+string pero no string+fecha o string+entero.

**07/11/2024 08:04 a.m. 156**

**Programa 4)**

**calculator.py**

print("calculator program")

first\_number = input("first number: ")

second\_number = input("second number: ")

print(int(first\_number) + int(second\_number))

print(first\_number + second\_number)

- En la segunda y tercera líneas se cargan los números tipeados por el usuario pero INPUT los toma como string, asi que para que se imprima la suma en consola cada variable asociada a los 2 números debe convertirse a entero con función INT para que print sume los números como sucede en la línea 4.

- En la línea 5 se unen los 2 strings que se imprimen y por tanto no se realiza la suma sino que se hace una unión de caracteres alfanuméricos (str) que se unen.

**07/11/2024 08:04 a.m. 225**

**Programa 5)**

**arguments1.py**

"""

example: # running "py arguments1.py 2023-01-01" # will produce the following output:

['arguments1.py', '2023-01-01']

"""

import sys

print(sys.argv)

print(sys.argv[0]) # program name

print(sys.argv[1]) # first arg

- Las 5 primeras líneas encerradas en 2 grupos de comillas dobles están usadas como comentarios porque hay desarrolladores que las

Usan así pero estrictamente hablando y más recomendado es darles el uso de strings multilínea que pueden tener internamente

Comillas simples o comillas dobles.

- La sexta línea con “import sys” importa la función sys que permite obtener información del sistema. Con el método arg se puede obtener

El nombre del programa .py ejecutado con py.exe o Python.exe y los argumentos tipeados posteriormente en una lista.

- La séptima línea con print(sys.argv) imprime la lista con los 2 valores. Cuando se ejecuta "py arguments1.py 2023-01-01" aparecerá en la consola ['arguments1.py', '2023-01-01'] con el nombre del programa como primer valor y el primer argumento como segundo valor.

- La octava línea con print(sys.argv[0]) imprime el nombre del programa 'arguments1.py'

- La novena línea con print(sys.argv[1]) imprime el nombre del primer argumento tipeado después de arguments1.py que en este ejemplo es: 2023-01-01

**07/11/2024 08:04 a.m. 134**

**Programa 6)**

**parsecs.py**

# Enter code below

parsecs = 11

lightyears = parsecs \* 3.26

print(str(parsecs) + " parsecs is " + str(lightyears) + " lightyears" )

- En este ejemplo hay 2 variables numéricas (parsecs y lightyears) y para poder imprimir en combinación con strings el comando print de

La línea 4 tiene que convertir las variables a strings con método STR para que los datos que se concatenan con PRINT sean todas del tipo STR.

**07/11/2024 08:04 a.m. 18**

**Programa 7)**

**program.py**

sum=1+2

print(sum)

- Ejemplo de print para imprimir una variable numérica a diferencia del ejemplo previo que imprime strings.

**07/11/2024 08:04 a.m. 172**

**Programa 8)**

**print2.py**

print("show this in console ")

"""

This script prints a message to the console.

Functions:

    None

Usage:

    Run this script to display a message in the console.

"""

- El comando print muestra en pantalla el string que tiene entre paréntesis y dobles comillas como otros ejemplos previos pero sin combinación

Con variables y concetenación sino con comillas dobles que podrían ser comillas simples.

- Se usan varias líneas entre comillas triples como comentarios para documentación dentro del propio programa (script) de Python.

**07/11/2024 08:04 a.m. 31**

**Programa 9)**

**print.py**

print("show this in console ")

- Script de python para impresión de str de consola sin comentarios del programa como el ejemplo anterior.

**07/11/2024 08:04 a.m. 454**

**Programa 10)**

**test1.py**

planets\_in\_solar\_system = 8 # int, pluto used to be the 9th planet, but is too small

print(f"planets in solar system = {planets\_in\_solar\_system}")

distance\_to\_alpha\_centauri = 4.367 # float, lightyears

print(f"distance to alpha centauri = {distance\_to\_alpha\_centauri}")

can\_liftoff = True

print(f"can liftoff = {can\_liftoff}")

shuttle\_landed\_on\_the\_moon = "Apollo 11" #string

print(f"shuttle landed on the moon = {shuttle\_landed\_on\_the\_moon}")

- Impresión de “f strings” disponible en PYTHON 3.X combinando strings con variables de cualquier tipo que se colocan con llaves dentro del PRINT empezando con f y dobles comillas que abren al comienzo del print y cierran al final del print cuando se combinan strings con variables de cualquier tipo entre llaves.

**07/11/2024 08:04 a.m. 51**

**Programa 11)**

**variabletest.py**

sum = 1 + 2 # 3

product = sum \* 2

print(product)

- Ejemplo de cálculo de 2 variables numéricas e impresión de un resultado.

**07/11/2024 08:04 a.m. 297**

**Programa 12)**

**test2.py**

distance\_to\_alpha\_centauri = 4.367 # looks like a float

type(distance\_to\_alpha\_centauri) ## <class 'float'>

typevar=type(distance\_to\_alpha\_centauri) ## <class 'float'>

print(typevar) ## <class 'float'>

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(type(distance\_to\_alpha\_centauri))

- Calculo e Impresión de impression de tipo de variable FLOAT en líneas 3,4 y 6

**07/11/2024 09:50 a.m. 85**

**Programa 13)**

**TeseEXPRESSION1.py**

a = 97

b = 55

# test expression

if a < b:

    # statement to be run

    print(b)

- Ejemplo de IF con condición resultante FALSE y por tanto no corre CODE BLOCK indentado.

**07/11/2024 09:52 a.m. 40**

**Programa 14)**

**TeseEXPRESSION2.py**

a = 93

b = 27

if a >= b:

    print(a)

- Programa con uso IF para chequeo de condición TRUE y ejecución de CODE BLOCK con comando print(a). El resultado de la condición del IF

es contrario al programa 13)

**07/11/2024 09:55 a.m. 54**

**Programa 15)**

**TeseEXPRESSION3.py**

a = 24

b = 44

if a <= 0:

    print(a)

    print(b)

- La condición del IF resulta False y por tanto no se ejecutan los 2 comandos print indentados que están en el CODE BLOCK.

**07/11/2024 09:56 a.m. 50**

**Programa 16)**

**TeseEXPRESSION4.py**

a = 24

b = 44

if a <= 0:

    print(a)

print(b)

- En el ejemplo anterior es parecido al programa 15 pero esta vez se ejecuta print(b) porque no está indentado y por tanto está fuera del CODE BLOCK.

**07/11/2024 09:57 a.m. 61**

**Programa 17)**

**TeseEXPRESSION5.py**

a = 27

b = 93

if a >= b:

    print(a)

else:

    print(b)

- Esta vez se ejecuta comando print(b) porque se usa ELSE en el IF y la condición da FALSE.

**07/11/2024 10:10 a.m. 113**

**Programa 18)**

**elif.py**

a = 27

b = 93

if a <= b:

    print("a is less than or equal to b")

elif a == b:

    print("a is equal to b")

- Consulta de varias condiciones con el uso de IF y ELIF

**07/11/2024 10:13 a.m. 116**

**Programa 19)**

**elif2.py**

a = 93

b = 27

if a <= b:

    print("a is less than or equal to b")

elif a > b:

    print("a is greater than b")

- Se ejecutá el code block de la segunda condición que aparece después de ELIF.

**07/11/2024 10:15 a.m. 142**

**Programa 20)**

**elif3.py**

a = 27

b = 93

if a < b:

    print("a is less than b")

elif a > b:

    print("a is greater than b")

else:

    print ("a is equal to b")

- Combinando IF, ELIF y ELSE puedo chequear 3 condiciones y ejecutar código en cada uno de los 3 casos.

**07/11/2024 10:15 a.m. 142**

**Programa 21)**

**elif4.py**

a = 93

b = 93

if a < b:

    print("a is less than b")

elif a > b:

    print("a is greater than b")

else:

    print ("a is equal to b")

- Combinando IF, ELIF y ELSE puedo chequear 3 condiciones y ejecutar código en cada uno de los 3 casos.

**07/11/2024 02:36 p.m. 262**

**Programa 22)**

**elif5.py**

a = 26

b = 25

c = 27

if a > b:

    if b > c:

        print ("a is greater than b and b is greater than c")

    else:

        print ("a is greater than b and less than c")

elif a == b:

    print ("a is equal to b")

else:

    print ("a is less than b")

- Uso de IF anidado y combinación con ELIF y ELSE del IF principal

**07/11/2024 02:37 p.m. 262**

**Programa 23)**

**elif6.py**

a = 26

b = 25

c = 27

if a > b:

    if b > c:

        print ("a is greater than b and b is greater than c")

    else:

        print ("a is greater than b and less than c")

elif a == b:

    print ("a is equal to b")

else:

    print ("a is less than b")

- Similar al programa anterior

**07/11/2024 02:39 p.m. 355**

**Programa 24)**

**elif7.py**

if test\_expression:

    # statement(s) to be run

    if test\_expression:

        # statement(s) to be run

    else:

        # statement(s) to be run

elif test\_expression:

    # statement(s) to be run

    if test\_expression:

        # statement(s) to be run

    else:

        # statement(s) to be run

else:

    # statement(s) to be run

- Este texto no es un programa que ejecuta sino que uestra la sintaxis de IF más ELIF con anidaciones IF y ELSE

**07/11/2024 02:56 p.m. 56**

**Programa 25)**

**OR\_operator.py**

a = 23

b = 34

if a == 34 or b == 34:

    print(a + b)

- Ejemplo de IF con condición que incluye OR OPERATOR y ejecuta code block porque el resultado de condición OR es True

**07/11/2024 02:59 p.m. 94**

**Programa 26)**

**And\_OPERATOR.py**

a = 23

b = 34

if a == 34 and b == 34:

    # nothing to print below - ALO

    print (a + b)

- Ejemplo de IF con condición que incluye AND OPERATOR y NO ejecuta code block porque el resultado de condición AND es False

**07/11/2024 03:03 p.m. 181**

**Programa 27)**

**And\_OPERATOR\_v2.py**

# Add code below

object\_size = 10

proximity = 9000

if object\_size > 5 and proximity < 1000:

    print('Evasive maneuvers required')

else:

    print('Object poses no threat')

- Ejemplo de IF con condición que incluye AND OPERATOR y ejecuta code block del ELSE porque el resultado de condición AND es False

**07/11/2024 03:03 p.m. 180**

**Programa 28)**

**And\_OPERATOR\_v3.py**

# Add code below

object\_size = 10

proximity = 900

if object\_size > 5 and proximity < 1000:

    print('Evasive maneuvers required')

else:

    print('Object poses no threat')

- Ejemplo de IF con condición que incluye AND OPERATOR y ejecuta code block inicial del IF porque el resultado de condición AND es True.

**07/11/2024 03:14 p.m. 111**

**Programa 29)**

**string-v1.py**

fact = "The Moon has no atmosphere."

two\_facts = fact + "No sound can be heard on the Moon."

print(two\_facts)

- Combinación de string directo de texto y variable de string en otra variable tipo string e impresión del resultado

**07/11/2024 03:28 p.m. 819**

**Programa 30)**

**string-v2.py**

"""

You can enclose Python strings in single, double, or triple quotation marks.

However, when a string contains words, numbers, or special characters (a substring) that are also enclosed in quotation marks,

 you should use a different style. For example, if a substring uses double quotation marks,

   enclose the entire string in single quotation marks, as shown here:

   'The "near side" is the part of the Moon that faces the Earth.'

   Similarly, if there are single quotation marks (or an apostrophe, as in Moon's in the following example) anywhere within the string,

     enclose the entire string in double quotation marks:

     "We only see about 60% of the Moon's surface."

"""

fact = "The Moon has no atmosphere."

two\_facts = fact + " No sound can be heard on the Moon."

print(two\_facts)

- Explicación en texto del archivo .py que está entre las líneas de 3 comillas de ejemplos del uso de comillas simples, dobles o triples en strings

**07/11/2024 03:33 p.m. 1.159**

**Programa 31)**

**string-v3.py**

"""

You can enclose Python strings in single, double, or triple quotation marks.

However, when a string contains words, numbers, or special characters (a substring) that are also enclosed in quotation marks,

 you should use a different style. For example, if a substring uses double quotation marks,

   enclose the entire string in single quotation marks, as shown here:

   'The "near side" is the part of the Moon that faces the Earth.'

   Similarly, if there are single quotation marks (or an apostrophe, as in Moon's in the following example) anywhere within the string,

     enclose the entire string in double quotation marks:

     "We only see about 60% of the Moon's surface."

 When the text has a combination of single and double quotation marks,

   you can use triple quotation marks to prevent problems with the interpreter

"""

fact = "The Moon has no atmosphere."

two\_facts = fact + " No sound can be heard on the Moon."

print(two\_facts)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

print("""We only see about 60% of the Moon's surface, this is known as the "near side".""")

- Ejemplo de impression de string con comillas triples que contiene texto con comillas simples y comillas dobles.

**07/11/2024 03:36 p.m. 100**

**Programa 32)**

**NewLineCharacter-v1.py**

multiline = "Facts about the Moon:\n There is no atmosphere.\n There is no sound."

print(multiline)

- Uso de \n para crear nueva línea en la impresión de un string.

**07/11/2024 03:37 p.m. 105**

**Programa 33)**

**NewLineCharacter-v2.py**

multiline = """Facts about the Moon:

 There is no atmosphere.

 There is no sound."""

print(multiline)

- Creación de string multilínea usando comillas triples, almacenamiento del str en una variable e impresión de la variable que contiene el string multilínea.

**07/11/2024 03:41 p.m. 152**

**Programa 34)**

**sTring\_method\_tittle-v1.py**

# observar como se convierte a MAYUSCULAS la primera letra de cada palabra con el metodo title()

print("temperatures and facts about the moon".title())

- observar como se convierte a MAYUSCULAS la primera letra de cada palabra con el método title()

**07/11/2024 03:42 p.m. 202**

**Programa 35)**

**sTring\_method\_tittle-v2.py**

# observar como se convierte a MAYUSCULAS la primera letra de cada palabra con el metodo title()

heading = "temperatures and facts about the moon"

heading\_upper = heading.title()

print(heading\_upper)

- observar como se convierte a MAYUSCULAS la primera letra de cada palabra con el método title(), crear una variable para guardar la conversión del método TITLE() e imprimir el nuevo resultado.

**07/11/2024 03:45 p.m. 152**

**Programa 36)**

**sTring\_method\_SPLIT-v1.py**

temperatures = "Daylight: 260 F Nighttime: -280 F"

temperatures\_list = temperatures.split()

print(temperatures\_list)

print(type(temperatures\_list))

- Usar metodo SPLIT() para convertir un string con palabras separadas por espacios en una lista de palabras que se pueden guardar en una variable. También se usa el método TYPE() para ver el tipo de objeto de la variable tipo LISTA que da como resultado CLASS LIST (clase tipo lista)

**07/11/2024 03:48 p.m. 158**

**Programa 37)**

**sTring\_method\_SPLIT- by NEWLINE character-v2.py**

temperatures = "Daylight: 260 F\n Nighttime: -280 F"

temperatures\_list = temperatures.split('\n')

print(temperatures\_list)

print(type(temperatures\_list))

- Ejemplo adicional de método SPLIT() pero usando \n como parámetro en vez del espacio que se usa por defecto para crear una lista donde cada elemento se crea según las ocurrencias de \n (que se usa como nueva línea en strings).

**07/11/2024 03:53 p.m. 237**

**Programa 37)**

**sTring\_method\_search\_with\_IN-v1.py**

print("Moon" in "This text will describe facts and challenges with space travel")

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

print("Moon" in "This text will describe facts about the Moon")

- El commando IN permite determinar una condición de un string si una palabra aparece en el texto posterior. En este código primero se imprime una línea con False porque la palabra MOON no aparece en la primera línea de PRINT, luego aparece la línea de asteriscos del segundo PRINT y el tercer PRINT dice True porque la palabra MOON aparece en el texto This text will describe facts about the Moon

**07/11/2024 03:55 p.m. 375**

**Programa 38)**

**sTring\_FIND-v1.py**

temperatures = """Saturn has a daytime temperature of -170 degrees Celsius, while Mars has -28 Celsius."""

print(temperatures.find("Moon"))

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

temperatures = """Saturn has a daytime temperature of -170 degrees Celsius, while Mars has -28 Celsius."""

print(temperatures.find("Mars"))

- El método FIND() refleja la posición que aparece una palabra buscada en un string y sino encuentra la palabra devuelve -1. En el ejemplo anterior el primer PRINT no consigue la palabra MOON y por eso FIND devuelve -1 y en el tercer print la palabra Mars si aparece y el FIND devuelve 64 (posición del carácter donde empieza MARS).

**07/11/2024 03:59 p.m. 273**

**Programa 38)**

**sTring\_method\_COUNT-v1.py**

temperatures = """Saturn has a daytime temperature of -170 degrees Celsius, while Mars has -28 Celsius."""

print(temperatures.count("Mars"))

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

print(temperatures.count("Moon"))

- El método COUNT() para strings cuenta el número de veces que una palabra aparece en un string. En el ejemplo el primer print devuelve 1 porque MARS aparece una vez y en el tercer PRINT devuelvo 0 porque MOON no aparece en el string.

**07/11/2024 04:01 p.m. 272**

**Programa 39)**

**sTring\_method\_LOWER-v1.py**

"""

Strings in Python are case-sensitive, which means that Moon and moon are considered different words.

To do a case-insensitive comparison, you can convert a string to all lowercase letters by using the .lower() method:

"""

print("The Moon And The Earth".lower())

- Strings in Python are case-sensitive, which means that Moon and moon are considered different words.

To do a case-insensitive comparison, you can convert a string to all lowercase letters by using the .lower() method

**07/11/2024 04:04 p.m. 181**

**Programa 40)**

**sTring\_method\_UPPER-v1.py**

"""

Similar to the .lower() method, strings have an .upper() method that does the opposite, converting every character to uppercase:

"""

print("The Moon And The Earth".upper())

- Similar to the .lower() method, strings have an .upper() method that does the opposite, converting every character to uppercase

**07/11/2024 04:13 p.m. 115**

**Programa 41)**

**sTring\_method\_CHECK\_CONTENT-v1.py**

temperatures = "Mars Average Temperature: -60 C"

parts = temperatures.split(':')

print(parts)

print(parts[-1])

- En este ejemplo se crea una lista de 2 elementos con el primer valor lo que aparece antes de los : del String porque los 2 puntos son el parámetro de separación del Split. El segundo valor de la lista es lo que viene después de los 2 puntos. Cuando se usa print(parts[-1]) se imprime el último (segundo) elemento de la lista que es “ -60 C”.

**07/11/2024 04:24 p.m. 262**

**Programa 42)**

**sTring\_method\_CHECK\_CONTENT\_isnumeric-v1.py**

mars\_temperature = "The highest temperature on Mars is about 30 C"

for item in mars\_temperature.split():

    print(item)

    print(type(item))

    if item.isnumeric():

        print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\* Si hay un al menos un NUMERO \*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

        print(item)

- Método ISNUMERIC() creo que devuelve True si aparecen números enteros en un string

**07/11/2024 04:30 p.m. 40 sTring\_method\_CHECK\_CONTENT\_isdecimal-v1.py**

**- ?**

**07/11/2024 04:32 p.m. 101**

**Programa 43)**

**sTring\_method\_STARSWITCH-v1.py**

# output   of the following instruction should be TRUE:  True

print("-60".startswith('-'))

- Metodo startswith() para verificar el texto del comienzo de un string. El ejemplo da True porque el str comienzo con guion ( - )

**07/11/2024 04:33 p.m. 69**

**sTring\_method\_ENDSWITCH-v1.py**

if "30 C".endswith("C"):

    print("This temperature is in Celsius")

- Metodo endswith() para verificar el texto del final de un string. En este ejemplo la condición da True y se ejecuta el comando print.

**07/11/2024 04:37 p.m. 340**

**Programa 44)**

**sTring\_method\_REPLACE-v1.py**

print("Saturn has a daytime temperature of -170 degrees Celsius, while Mars has -28 Celsius.")

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

print("Saturn has a daytime temperature of -170 degrees Celsius, while Mars has -28 Celsius.".replace("Celsius", "C"))

- el método REPLACE() cambia un string. En este ejemplo se cambia la descripción de grados de temperatura de CELSIUS a C.

**07/11/2024 04:41 p.m. 315**

**Programa 45)**

**sTring\_method\_JOIN-v1.py**

"""

The .join() method requires an iterable (such as a Python list) as an argument, so its usage looks different from other string methods:

"""

moon\_facts = ["The Moon is drifting away from the Earth.", "On average, the Moon is moving about 4cm every year."]

print(moon\_facts)

print(' '.join(moon\_facts))

- El método JOIN() une strings (no lo he revisado bien).

**07/11/2024 04:49 p.m. 120**

**Programa 46)**

**sTring\_with-VARIABLES-v1.py**

mass\_percentage = "1/6"

print("On the Moon, you would weigh about %s of your weight on Earth." % mass\_percentage)

- Otra forma de combinar textos con variables en string. Prefiero f” strings.

**07/11/2024 04:54 p.m. 205**

**Programa 47)**

**sTring\_with-MULTIPLE\_VALUES-v1.py**

print("""Both sides of the %s get the same amount of sunlight, but only one side is seen from %s because the %s rotates around its own axis when it orbits %s.""" % ("Moon", "Earth", "Moon", "Earth"))

- Otra forma de cambiar múltiples valores en strings

**07/11/2024 04:57 p.m. 120**

**Programa 48)**

**sTring\_method\_FORMAT-v1.py**

mass\_percentage = "1/6"

print("On the Moon, you would weigh about {} of your weight on Earth.".format(mass\_percentage))

- Sustitución o incorporación de valores en string con FORMAT() método, que es también parecido como una sustitución de variable llamando una función con un parámetro para cambiar a un string.

**07/11/2024 04:59 p.m. 167**

**Programa 49)**

**sTring\_method\_FORMAT-v2.py**

mass\_percentage = "1/6"

print("""You are lighter on the {0}, because on the {0} you would weigh about {1} of your weight on Earth.""".format("Moon", mass\_percentage))

- Cambio de string con FORMAT() method usando los valores posicionales de una lista.

**07/11/2024 05:03 p.m. 365**

**Programa 50)**

**sTring\_method\_F-STRINGS-v1.py**

mass\_percentage = "1/6"

print("""You are lighter on the {moon}, because on the {moon} you would weigh about {mass} of your weight on Earth.""".format(moon="Moon", mass=mass\_percentage))

print(f"On the Moon, you would weigh about {mass\_percentage} of your weight on Earth.")

print(f"On the Moon, you would weigh about {round(100/6, 1)}% of your weight on Earth.")

- Uso de F-Strings que son los que más me gustan y en la última línea se usa método ROUND() para redondear una división usando un decimal.

**07/11/2024 05:15 p.m. 285**

**Programa 51)**

**sTring\_method\_F-STRINGS-v2.py**

"""

Using an expression doesn't require a function call. Any of the string methods are valid as well.

 For example, the string could enforce a specific casing for creating a heading:

"""

subject = "interesting facts about the moon"

heading = f"{subject.title()}"

print(heading)

**- Uso de la función TITLE() para convertir en mayúscula la primera letra de cada palabra de un string.**

**07/11/2024 05:22 p.m. 266**

**Programa 52)**

**sTring\_methods\_several-v1.py**

name = 'Ganymede'

planet = 'Mars'

gravity = '1.43'

template = """Gravity Facts about {name}

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

Planet Name: {planet}

Gravity on {name}: {gravity} m/s2"""

print(template.format(name=name, planet=planet, gravity=gravity))

- Otro ejemplo de usar método FORMAT() combinado con el uso de variables y strings multilínea con uso de comillas triplicadas.

**07/11/2024 05:54 p.m. 33**

**Programa 53)**

**math\_ADDITION.py**

answer = 30 + 12

print(answer)

- Calculo de suma y almacenamiento en variable numérica que después se puede imprimir.

**07/11/2024 05:55 p.m. 41**

**Programa 54)**

**math\_SUBSTRACTION.py**

difference = 30 - 12

print(difference)

- Calculo de RESTA y almacenamiento en variable numérica que después se puede imprimir.

**07/11/2024 05:55 p.m. 35**

**Programa 55)**

**math\_MULTIPLICATION.py**

product = 30 \* 12

print(product)

- Calculo de MULTIPLICACIÓN y almacenamiento en variable numérica que después se puede imprimir.

**07/11/2024 05:56 p.m. 37**

**Programa 56)**

**math\_DIVISION.py**

quotient = 30 / 12

print(quotient)

- Calculo de DIVISIÓN y almacenamiento en variable numérica que después se puede imprimir.

**07/11/2024 05:57 p.m. 70**

**Programa 57)**

**math\_FLOOR\_DIVISION.py**

seconds = 1042

display\_minutes = 1042 // 60

print(display\_minutes)

- Calculo de division que sólo muestra la parte entera del resultado sin los decimales (creo).

**07/11/2024 05:58 p.m. 69**

**Programa 58)**

**math\_DIVISION-V2.py**

seconds = 1042

display\_minutes = 1042 / 60

print(display\_minutes)

- Calculo de division normal que muestra decimales.

**07/11/2024 06:00 p.m. 125**

**Programa 59)**

**math\_MODULE-V1.py**

seconds = 1042

display\_minutes = 1042 // 60

display\_seconds = 1042 % 60

print(display\_minutes)

print(display\_seconds)

**07/11/2024 06:03 p.m. 176**

**Programa 60)**

**math\_OPERATION\_ORDER-V1.py**

"""

PRIORITY ORDER OF OPERATIONS

1) Parentheses

2) Exponents

3) Multiplication and division

4) Addition and subtraction

"""

result\_2 = 1032 + (26 \* 2)

print(result\_2)

- Prioridad de operaciones matemáticas: 1) Parentheses. 2) Exponents. 3) Multiplication and division. 4) Addition and subtraction

**07/11/2024 06:07 p.m. 175**

**Programa 61)**

**math\_OPERATIONs\_examples-V1.py**

first\_planet = 149597870

second\_planet = 778547200

distance\_km = second\_planet - first\_planet

print(distance\_km)

distance\_mi = distance\_km / 1.609344

print(distance\_mi)

- 2 ejemplos de resta y división

**07/11/2024 06:08 p.m. 90**

**Programa 62)**

**math\_STRING\_TO\_NUMBERS-V1.py**

demo\_int = int('215')

print(demo\_int)

demo\_float = float('215.3')

print(demo\_float)

- Métodos INT() Y float() para convertir strings a números enteros o con decimales (punto flotante).

**07/11/2024 06:09 p.m. 74**

**Programa 63)**

**math\_aboslute\_values-V1.py**

print(39 - 16)

print(16 - 39)

print(abs(39 - 16))

print(abs(16 - 39))

- Uso de función abs() para valor absoluto (siempre positivo).

**07/11/2024 06:11 p.m. 74**

**Programa 64)**

**math\_ROUNDING-V1.py**

print(round(1.4))

print(round(1.5))

print(round(2.5))

print(round(2.6))

- Metodo round() para redondeo dependiendo del valor decimal

**07/11/2024 06:14 p.m. 119**

**Programa 65)**

**math\_LIBRARY-V1.py**

from math import ceil, floor

round\_up = ceil(12.5)

print(round\_up)

round\_down = floor(12.5)

print(round\_down)

- Importación de funciones ceil() y floor() del módulo math (o librería math) para redondeo de decimales hacia arriba o hacia abajo.

**07/11/2024 06:23 p.m. 335**

**Programa 66)**

**math\_example-v1.py**

first\_planet\_input = input('Enter the distance from the sun for the first planet in km ')

second\_planet\_input = input('Enter the distance from the sun for the second planet in km ')

first\_planet = int(first\_planet\_input)

second\_planet = int(second\_planet\_input)

distance\_km = second\_planet - first\_planet

print(abs(distance\_km))

- En este ejemplo se usa INPUT() para leer datos de usuario en modo string, luego se usa INT() para convertir string (STR) a valor entero

Y finalmente función abs() para obtener valor absoluto (positivo) de un resultado entero positivo o negativo dependiendo del valor de las variables first\_planet y second\_planet

**07/11/2024 06:32 p.m. 350**

**Programa 67)**

**LISTS-example-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

print("The first planet is", planets[0])

print("The second planet is", planets[1])

print("The third planet is", planets[2])

print("The LAST  planet is", planets[-1])

- En este ejemplo se muestran los valores resultantes de los índices de una lista empezando por índice 0 para para el primer planeta,

Índice 1 para el segundo planeta y así sucesivamente con el caso especial que el último planeta siempre se puede referenciar con el

Índice -1.

**07/11/2024 06:34 p.m. 162**

**Programa 68)**

**LISTS-example-V2.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets[3] = "Red Planet"

print("Mars is also known as", planets[3])

- En este ejemplo se cambia el cuarto planeta de la lista (índice 3) sustiyendo el string “Mars” por el string “Red Planet”.

**07/11/2024 06:35 p.m. 194**

**Programa 69)**

**LISTS-LENGHT\_of\_a\_list-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

number\_of\_planets = len(planets)

print("There are", number\_of\_planets, "planets in the solar system.")

- En este ejemplo se usa la función o método len() para devolver el número 8 que es el número de elementos de la LISTA y también

Obsérvense los espacios de string agregados en el comando print() antes y después de las comas.

**07/11/2024 07:03 p.m. 673**

**Programa 70)**

**LISTS-APPEND\_or\_ADD\_to\_a\_list-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets.append("Pluto")

number\_of\_planets = len(planets)

print("There are actually", number\_of\_planets, "planets in the solar system.")

print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

# for i, planet in enumerate(planets, start=1):

#    print(f"planeta {i} es {planet}")

# print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

for contador in planets:

    print(contador)

    indice=planets.index(contador)

    print(indice)

    print(f"planeta es {contador}")

- Este ejemplo es muy bueno. Se usá el método append(“Pluto”) para agregar un noveno planeta a la LISTA de planetas. Ahora el

Method() que esta vez devuelve 9 después del append() a decir del ejemplo anterior de la lista. – Las líneas en verde muestra comentarios

con # al comienzo de la línea que no recuerdo para que son. – Luego se usa un lazo con FOR con CONTADOR como variable del lazo

que va recomiendo toda la lista. – También se imprimen los índices empezando desde 0 con print(indice) y print(contador) para ver los valores

de la lista (los 9 planetas después del append() para agregar a PLUTO) y también la última línea con print combina texto en F-STRINGS con los valores de la lista que aparecen sequencialmente por el FOR con {contador}

**07/11/2024 07:06 p.m. 686**

**Programa 70)**

**LISTS-INDEX\_EXAMPLE-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets.append("Pluto")

number\_of\_planets = len(planets)

print("There are actually", number\_of\_planets, "planets in the solar system.")

print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

# for i, planet in enumerate(planets, start=1):

#    print(f"planeta {i} es {planet}")

# print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

for contador in planets:

    print(contador)

    indice=planets.index(contador) + 1

    print(indice)

    print(f"planeta {indice} es {contador}")

- Parecido al ejemplo anterior pero se suma 1 al índice para imprimir en la última línea reflejando el número de planeta de 1 a 9 sin usar

El valor del índice 0.

**07/11/2024 07:07 p.m. 689**

**Programa 71)**

**LISTS-INDEX\_EXAMPLE-V2.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets.append("Pluto")

number\_of\_planets = len(planets)

print("There are actually", number\_of\_planets, "planets in the solar system.")

print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

# for i, planet in enumerate(planets, start=1):

#    print(f"planeta {i} es {planet}")

# print("     \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*          ")

for contador in planets:

    print(contador)

    indice=planets.index(contador) + 1

    print(indice)

    print(f"planeta {indice} es {contador} \n")

- Similar al ejemplo anterior pero se agrega NEW LINE con \n para cada bloque de ejecución del FOR para mostrar una separación.

**07/11/2024 08:01 p.m. 253**

**Programa 72)**

**LISTS-REMOVE\_last\_element\_from\_a\_list-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"]

planets.pop()  # Goodbye, Pluto

number\_of\_planets = len(planets)

print("No, there are definitely", number\_of\_planets, "planets in the solar system.")

- Se usó metodo pop() para borrar el último elemento de la lista.

**07/11/2024 08:05 p.m. 479**

**Programa 73)**

**LISTS-CERO\_OR\_NEGATIVE\_INDEXES-V1.py**

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"]

planets.pop()  # Goodbye, Pluto

number\_of\_planets = len(planets)

print("No, there are definitely", number\_of\_planets, "planets in the solar system.")

print("The first planet is", planets[0])

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

print("The last planet is", planets[-1])

print("The penultimate planet is", planets[-2])

- Se usan elementos negativos de la lista para usar índices del final de la lista y retroceder. Con planets[-1] se puede acceder el valor del

Último planeta y con planets[-2] el penúltimo planeta (segundo elemento del final hacia atrás).

**07/11/2024 08:10 p.m. 484**

**Programa 74)**

**LISTS-FIND\_A\_VALUE-V1.py**

#  Because indexing starts with 0, you need to add 1 to display the proper number.

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

jupiter\_index = planets.index("Jupiter")

print("Jupiter is the", jupiter\_index + 1, "planet from the sun")

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

mercury\_index = planets.index("Mercury")

print("Mercury is the", mercury\_index + 1, "planet from the sun")

- Otro ejemplo de índices pero buscando un valor en la tabla y determinando su índice con index(“Jupiter”) para encontrar el índice

Del planeta Jupiter en la lista.

**07/11/2024 08:23 p.m. 792**

**Programa 75)**

**LISTS-MAX\_and\_MIN-V1.py**

gravity\_on\_earth = 1.0

gravity\_on\_the\_moon = 0.166

gravity\_on\_planets = [0.378, 0.907, 1, 0.377, 2.36, 0.916, 0.889, 1.12]

gravity\_on\_planets = [0.378, 0.907, 1, 0.377, 2.36, 0.916, 0.889, 1.12]

bus\_weight = 124054 # in Newtons, on Earth

print("On Earth, a double-decker bus weighs", bus\_weight, "N")

print("On Mercury, a double-decker bus weighs", bus\_weight \* gravity\_on\_planets[0], "N")

gravity\_on\_planets = [0.378, 0.907, 1, 0.377, 2.36, 0.916, 0.889, 1.12]

bus\_weight = 124054 # in Newtons, on Earth

print("On Earth, a double-decker bus weighs", bus\_weight, "N")

print("The lightest a bus would be in the solar system is", bus\_weight \* min(gravity\_on\_planets), "N")

print("The heaviest a bus would be in the solar system is", bus\_weight \* max(gravity\_on\_planets), "N"

- En este ejemplo se usa método o función min() para encontrar el mínimo valor numérico de la lista y max() para encontrar el mínimo valor numérico de la lista. Con esas funciones las últimas 2 líneas hacen cálculos multiplicando el valor relativo de la gravedad de los planetas

(gravity\_on\_planets) por el peso del bus (bus\_weight) para imprimir los pesos de bus más liviano y pesado que aparecerían en los planetas

Del sistema solar.

**07/11/2024 08:28 p.m. 353**

**Programa 76)**

**LISTS-slice lists-V1.py**

# A slice creates a new list. It doesn't modify the current list.

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune"]

planets\_before\_earth = planets[0:2]

print(planets\_before\_earth)

planets\_after\_earth = planets[3:7]

print(planets\_after\_earth)

planets\_after\_earth = planets[3:]

print(planets\_after\_earth)

planets\_after\_earth = planets[3:8]

print(planets\_after\_earth)

- Se usan rangos de indices para seleccionar algunos valores y crear una nueva lista: planets\_before\_earth = planets[0:2] crea lista con

2 valores usando índice 0 e indice 1 (no se incluye el 2). - planets\_after\_earth = planets[3:7] crea lista con 4 valores usando índices 3 al 6. - planets\_after\_earth = planets[3:] crea lista con 5 valores usando índices 3 al último índice. - planets\_after\_earth = planets[3:8] crea lista con 5 valores usando índices 3 al 7 (no se incluye el 8).

**07/11/2024 08:29 p.m. 259**

**Programa 77)**

**LISTS-JOIN lists-V1.py**

amalthea\_group = ["Metis", "Adrastea", "Amalthea", "Thebe"]

galilean\_moons = ["Io", "Europa", "Ganymede", "Callisto"]

regular\_satellite\_moons = amalthea\_group + galilean\_moons

print("The regular satellite moons of Jupiter are", regular\_satellite\_moons)

- Se unen 2 listas usando el “ + “

**07/11/2024 08:33 p.m. 511**

**Programa 78)**

**LISTS-SORT lists-V1.py**

# To sort a list, use the .sort() method on the list.

#  Python sorts a list of strings in alphabetical order and a list of numbers in numeric order:

amalthea\_group = ["Metis", "Adrastea", "Amalthea", "Thebe"]

galilean\_moons = ["Io", "Europa", "Ganymede", "Callisto"]

regular\_satellite\_moons = amalthea\_group + galilean\_moons

print(regular\_satellite\_moons)

regular\_satellite\_moons.sort()

print(regular\_satellite\_moons)

print("The regular satellite moons of Jupiter are", regular\_satellite\_moons)

- Con el método SORT() se ordena una lista. EN el caso de strings se ordena la lista alfabéticamente en forma ascendente (a-z).

**07/11/2024 08:57 p.m. 132**

**Programa 79)**

**WHILE\_loop\_v1'.py**

user\_input = ''

while user\_input.lower() != 'done':

    user\_input = input('Enter a new value, or done when done ')

**- Mientras la condición del WHILE sea true se ejecuta “code block” indentado dentro del while**

**07/11/2024 09:03 p.m. 442**

**Programa 80)**

**WHILE\_and\_LIST\_creation\_v1.py**

# Create the variable for user input

user\_input = ''

# Create the list to store the values

inputs = []

# The while loop

while user\_input.lower() != 'done':

    # Check if there's a value in user\_input

    if user\_input:

        # Store the value in the list

        inputs.append(user\_input)

    # Prompt for a new value

    user\_input = input('Enter a new value, or done when done ')

# Print the list

print(inputs)

- Se inicializa user\_input como STR sin caracteres. – Se crea lista inputs sin valores. – Se crea una lista con los datos tipeados por el usuario

Usando método append() hasta que usuario escriba ‘done’.

**07/11/2024 09:11 p.m. 101**

**Programa 81)**

**FOR\_loops\_v1.py**

countdown = [4, 3, 2, 1, 0]

for number in countdown:

    print(number)

print("Blast off!! 🚀")

- Se combina un FOR con el recorrido de los elementos de una lista para ejecutar “code block” print(number)

**07/11/2024 09:12 p.m. 158**

**Programa 82)**

**FOR\_loops\_v2.py**

from time import sleep

countdown = [4, 3, 2, 1, 0]

for number in countdown:

    print(number)

    sleep(1)  # Wait 1 second

print("Blast off!! 🚀")

- Se combina el ejemplo anterior con función sleep(1) para esperar un segundo por cada iteración del FOR.

**13/11/2024 11:55 a.m. 4.131**

**Programa 83)**

**DataDictionary.py**

"""

Python uses curly braces ({ }) and the colon (:) to denote a dictionary.

 You can either create an empty dictionary and add values later, or populate it at creation time.

   Each key/value is separated by a colon, and the name of each key is contained in quotes as a string literal.

     Because the key is a string literal, you can use whatever name is appropriate to describe the value.

Let's create a dictionary to store the name of the planet Earth, and the number of moons Earth has:

"""

planet = {

    'name': 'Earth',

    'moons': 1

}

# Accessing Values in a Dictionary

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

#  https://learn.microsoft.com/en-us/training/modules/python-dictionaries/2-dictionary-basics

#

print(planet.get('name'))

print(planet.get('moons'))

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

# planet['name'] is identical to using planet.get('name')

print(planet['name'])

print(planet['moons'])

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

"""

Although the behavior of get and the square brackets ([ ]) is generally the same for retrieving items, there's one key difference.

 If a key isn't available, get returns None, and [ ] raises a KeyError.

wibble = planet.get('wibble') # Returns None

wibble = planet['wibble'] # Throws KeyError

"""

wibble = planet.get('wibble') # Returns None

print("""  wibble = planet['wibble'] # Throws KeyError """)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

"""

Modify dictionary values

You can also modify values inside a dictionary object, by using the update method. This method accepts a dictionary as a parameter,

 and updates any existing values with the new ones you provide.

 If you want to change the name for the planet dictionary, you can use the following code, for example:

planet.update({'name': 'Makemake'})

# No output: name is now set to Makemake.

"""

planet.update({'name': 'Makemake'})

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

planet.update({

    'name': 'Jupiter',

    'moons': 79

})

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

planet['name'] = 'Jupiter'

planet['moons'] = 79

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

planet['orbital period'] = 4333

# planet dictionary now contains: {

#   name: 'jupiter'

#   moons: 79

#   orbital period: 4333

# }

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

"""

To remove a key, you use pop. pop returns the value and removes the key from the dictionary.

 To remove orbital period, you can use the following code:

 planet.pop('orbital period')

# planet dictionary now contains: {

#   name: 'jupiter'

#   moons: 79

# }

"""

planet.pop('orbital period')

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

"""

Complex data types - NESTED DICTIONARIES

You can also nest dictionaries inside other dictionaries.

 For example, you can create a dictionary that contains a dictionary for each planet in our solar system.

 # Add address

planet['diameter (km)'] = {

    'polar': 133709,

    'equatorial': 142984

}

# planet dictionary now contains: {

#   name: 'Jupiter'

#   moons: 79

#   diameter (km): {

#      polar: 133709

#      equatorial: 142984

#   }

# }

"""

 # Add address

planet['diameter (km)'] = {

    'polar': 133709,

    'equatorial': 142984

}

print(planet)

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

print(f'{planet["name"]} polar diameter: {planet["diameter (km)"]["polar"]}')

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

- Ver los comentarios de este ejemplo en las líneas que comienzan con # o las múltiples líneas encerradas entre triples comillas dobles (“”’”).

**13/11/2024 12:05 p.m. 449**

**Programa 84)**

**DataDictionary-v2.py**

# Enter code below

planet = {

    'name': 'Mars',

    'moons': 2

}

print(planet)

# Output: {'name': 'Mars', 'moons': 2}

print(f'{planet["name"]} has {planet["moons"]} moon(s)')

# Output: Mars has 2 moon(s)

planet['circumference (km)'] = {

    'polar': 6752,

    'equatorial': 6792

}

print(f'{planet["name"]} has a polar circumference of {planet["circumference (km)"]["polar"]}')

# Output: Mars has a polar circumference of 6752

- Este es otro ejemplo que muestra DICCIONARIOS DE DATOS ANIDADOS (nested data dictionary).

**13/11/2024 12:30 p.m. 2.175**

**Programa 85)**

**DataDictionary-v3.py**

"""

Dynamic programming with dictionaries

In your program, you want to perform various calculations, like totaling the number of moons.

Additionally, as you get into more advanced programming, you might find that you're loading this type of information from files or

 a database, rather than coding directly into Python.

To help support these scenarios, Python enables you to treat both the keys and values inside of a dictionary as a list.

You can dynamically determine keys and values, and perform various calculations.

Imagine a dictionary storing monthly rainfall amounts. You would likely have keys for each month and the associated rainfall.

 You want to add up the total rainfall, and writing the code to perform the operation by using each individual key would be

   rather tedious.

   Retrieve all keys and values

The keys() method returns a list object that contains all the keys. You can use this method to iterate through

all items in the dictionary.

"""

rainfall = {

    'october': 3.5,

    'november': 4.2,

    'december': 2.1

}

for key in rainfall.keys():

    print(f'{key}: {rainfall[key]}cm')

"""

Determine if a key exists in a dictionary

if 'december' in rainfall:

    rainfall['december'] = rainfall['december'] + 1

else:

    rainfall['december'] = 1

# Because december exists, the value will be 3.1

"""

if 'december' in rainfall:

    print(f"rainfall in december is " + str(rainfall['december']))

    rainfall['december'] = rainfall['december'] + 1

    print(" one '(1)' was added to december rainfall ")

else:

    rainfall['december'] = 1

# Because december exists, the value will be 3.1

"""

Retrieve all values

Similar to keys(), values() returns the list of all values in a dictionary without their respective keys.

 values() can be helpful when you're using the key for labeling purposes, such as the preceding example,

 in which the keys are the name of the month. You can use values() to determine the total rainfall amount:

 """

total\_rainfall = 0

for value in rainfall.values():

    total\_rainfall = total\_rainfall + value

print(f'There was {total\_rainfall}cm in the last quarter.')

- Ejemplo de manejo de DATADICTIONARY como una lista de pares KEY-VALUE similares o con el mismo propósito como un diccionario de datos con una lista de KEY-VALUE que almacene como KEY cada mes del año y como VALUE el nivel de lluvia mensual. – Combinando un FOR con el método KEYS() se puede acceder a los diferentes valores de del diccionario de datos como que se estuviera recorriendo una lista con variables.

**13/11/2024 01:04 p.m. 1.802**

**Programa 86)**

**DataDictionary-v4.py**

planet\_moons = {

    'mercury': 0,

    'venus': 0,

    'earth': 1,

    'mars': 2,

    'jupiter': 79,

    'saturn': 82,

    'uranus': 27,

    'neptune': 14,

    'pluto': 5,

    'haumea': 2,

    'makemake': 1,

    'eris': 1

}

print(planet\_moons)

"""

Obtain a list of moons and number of planets

Python dictionaries allow you to retrieve all the values and keys by using the values and keys methods, respectively.

 Each method returns a list containing the data, which can then be used like a regular Python list. You can determine the number

   of items by using len, and iterate through it by using for loops. In the dictionary you created, the planet names are keys

   and the number of moons are the values.

Start by retrieving a list with the number of moons, and store this in a variable named moons. Then obtain the total number of planets and store that value in a variable named total\_planets.

"""

moons = planet\_moons.values()

print(planet\_moons.keys())

print(moons)

total\_planets = len(planet\_moons.keys())

print(f" total planets:  {total\_planets}")

"""

Determine the average number of moons

You will finish this exercise by determining the average number of moons. Start by creating a variable named total\_moons;

 this will be your counter for the total number of moons. Then add a for loop to loop through the list of moons, adding each value to total\_moons.

Finally, calculate the average by dividing total\_moons by total\_planets and displaying the value.

"""

# Each planet has an average of 17.833333333333332 moons

total\_moons = 0

for moon in moons:

    total\_moons = total\_moons + moon

average = total\_moons / total\_planets

print(f'Each planet has an average of {average} moons')

# Each planet has an average of 17.833333333333332 moons

- En este ejemplo se usa el método KEYS() de diccionario de datos para obtener una lista de keys y el método VALUES() de diccionario de datos para obtener una lista de los valores asociados a los KEY-VALUE PAIRS. – Se usa también método len() para contar el número de planetas. También se usa un FOR para calcular el número total de lunas sumando los valores de una lista.

**13/11/2024 03:12 p.m. 2.300**

**Programa 87)**

**functions-v1 - copia.py**

"""

Functions with no arguments

To create a function, you use the def keyword followed by a name, parentheses, and then the body with the function code:

def rocket\_parts():

    print("payload, propellant, structure")

    In this case, rocket\_parts is the name of the function. That name is followed by the empty parentheses,

    which indicate that no arguments are needed. Last is the code, indented with four spaces.

    To use the function, you must call it by its name by using parentheses:

"""

def rocket\_parts():

    print("payload, propellant, structure")

rocket\_parts()

"""

The rocket\_parts() function doesn't take any arguments and prints a statement.

If you need to use a value that a function is returning, you can assign the function output to a variable:

"""

output = rocket\_parts()

print(f"printing variable output: {output}")

# None

"""

The line output is None is a conditional statement in Python that checks whether the variable output

 is currently set to None. In Python,

   None is a special constant that represents the absence of a value or a null value.

     It is often used to signify that a variable has not been initialized or that it explicitly has no value.

     The is keyword in this statement is used for identity comparison. This means that output is None checks

       if output and None are the same object in memory. This is different from using the == operator,

       which checks for value equality rather than identity. Using is for comparison with None is the

         recommended practice in Python because None is a singleton, meaning there is only one

           instance of None in a Python runtime.

This type of check is common in Python code to determine if a variable has been assigned a meaningful

value or if it remains in its default, uninitialized state. For example, you might see this check

before performing operations that require output to have a valid value, ensuring that the code does

 not run into errors due to output being None. This helps in writing robust and error-free code by

   handling cases where a variable might not have been set to a proper value before its usage.

"""

question1 = output is None

print(question1)

print(" \*\*\*\*\*\*\*\*\*\*\*\* ")

print(output is None)

# True

- Creación de FUNCIONES sin argumentes con DEF y el retorno de None como constante especial cuando no hay valor que devolver en la función. Ver en este ejemplo

Y en los ejemplos previos recientes comentarios con # y comillas dobles usadas 3 veces (triplicadas).

**13/11/2024 03:12 p.m. 2.300**

**Programa 88)**

**functions-v1.py**

- Igual al anterior programa.

**13/11/2024 03:25 p.m. 1.409**

**Programa 89)**

**functions-v2.py**

"""

Required and optional arguments

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

In Python, several built-in functions require arguments. Some built-in functions make arguments optional.

Built-in functions are immediately available, so you don't need to import them explicitly.

An example of a built-in function that requires an argument is any(). This function takes an iterable (for example, a list)

 and returns True if any item in the iterable is True. Otherwise, it returns False.

"""

print(any([True, False, False]))

print(any([False, False, False]))

# The any() function returns True if any item in the iterable is True. Otherwise, it returns False.

# The any() function is a built-in function in Python that requires an argument.

# the following line of code IF EXECUTED  will raise an error because the any() function requires an argument.

#

# any()

#

"""

You can verify that some functions allow the use of optional arguments by using another built-in function called str().

 This function creates a string from an argument. If no argument is passed in, it returns an empty string:

"""

str()

# The str() function is a built-in function in Python that allows the use of optional arguments.

print(str())

# The str() function returns an empty string when no argument is passed in.

str(15)

# The str() function returns a string representation of the number 15.

print(str(15))

- Función ANY() toma algo iterable como una LISTA y devuelve True si algún elemento en el iterable es True. – La función STR() devuelve string de la conversión

De un número. – Ver los comentarios que hablan de argumentos REQUERIDOS y OPCIONALES.

**13/11/2024 03:25 p.m. 1.409**

**Programa 90)**

**functions-v2 - copia.py**

- Igual al anterior programa.

**13/11/2024 03:34 p.m. 1.173**

**Programa 91)**

**functions-v3.py**

"""

Use function arguments in Python

Now that you know how to create a function with no inputs, the next step is to create functions that require an argument.

Using arguments makes functions more flexible, because they can do more and conditionalize what they do.

Requiring an argument

If you're piloting a rocket ship, a function without required inputs is like a computer with a button to tell you the time.

If you press the button, a computerized voice tells you the time. But a required input can be a destination to calculate travel distance.

Required inputs are called arguments to the function.

To require an argument, put it within the parentheses:

"""

def distance\_from\_earth(destination):

    if destination == "Moon":

        return "238,855"

    else:

        return "Unable to compute to that destination"

# distance\_from\_earth()

# Output: TypeError: distance\_from\_earth() missing 1 required positional argument: 'destination'

distance\_from\_earth("Moon")

# Output: '238,855'

print(distance\_from\_earth("Moon"))

distance\_from\_earth("Saturn")

# Output: 'Unable to compute to that destination'

print(distance\_from\_earth("Saturn"))

- Uso de funciones con PARÁMETROS (también llamados ARGUMENTOS) combinados con condicionales para controlar lo

Que devuelve la función con RETURN. Cuando se usan parámetros REQUERIDOS y se omiten los mismos llamando la función aparecen errores.

**13/11/2024 03:34 p.m. 1.173**

**Programa 92)**

**functions-v3 - copia.py**

- Igual al anterior programa.

**13/11/2024 03:47 p.m. 798**

**Programa 93)**

**functions-v4 - copia.py**

- Igual al programa v4.

**13/11/2024 03:47 p.m. 798**

**Programa 94)**

**functions-v4.py**

"""

Multiple required arguments

To use multiple arguments, you must separate them by using a comma.

 Let's create a function that can calculate how many days it takes to reach a destination, given distance and a constant speed:

"""

def days\_to\_complete(distance, speed):

    hours = distance/speed

    return hours/24

days\_to\_complete(238855, 75)

print(days\_to\_complete(238855, 75))

"""

Functions as arguments

You can use the value of the days\_to\_complete() function and assign it to a variable, and then pass it to round()

(a built-in function that rounds to the closest whole number) to get a whole number:

"""

total\_days = days\_to\_complete(238855, 75)

round(total\_days)

print(round(total\_days))

round(days\_to\_complete(238855, 75))

print(round(days\_to\_complete(238855, 75)))

- Uso de multiples parámetros requeridos en funciones separados con coma al definir función DEF. Tambi’en se usa m’etodo de redondeo con ROUND(). En este ejemplo tambi’en se usan FUNCIONES como ARGUMENTOS, según entiendo, funciones que llaman a funciones.

**13/11/2024 04:00 p.m. 1.171**

**Programa 95)**

**functions-v5 - copia.py**

- Igual al programa v5.

**13/11/2024 04:00 p.m. 1.171**

**Programa 96)**

**functions-v5.py**

"""

Work with arguments in functions

Required arguments in functions are used when functions need those arguments to work properly.

In this exercise, you'll construct a fuel report that requires information from several fuel locations throughout the rocket ship.

"""

def generate\_report(main\_tank, external\_tank, hydrogen\_tank):

    output = f"""Fuel Report:

    Main tank: {main\_tank}

    External tank: {external\_tank}

    Hydrogen tank: {hydrogen\_tank}

    """

    print(output)

generate\_report(80, 70, 75)

"""

Use keyword arguments in Python - OPTIONAL ARGUMENTS

Optional arguments in functions are used when functions can work without them.

Keyword argument values must be defined in the functions themselves.

 When you're calling a function that's defined with keyword arguments, it isn't necessary to use them at all.

"""

from datetime import timedelta, datetime

def arrival\_time(hours=51):

    now = datetime.now()

    arrival = now + timedelta(hours=hours)

    return arrival.strftime("Arrival: %A %H:%M")

print(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ")

arrival\_time()

print(arrival\_time())

arrival\_time(hours=0)

print(arrival\_time(hours=0))

- Asignación de un F-STRING multilínea a una variable. – Uso de funciones con varios parámetros requeridos separados por coma. – Establecimiento de asignación

De valor por defecto a parámetros: ejemplo: def arrival\_time(hours=51): Uso de método *strftime* (ejemplo: return arrival.strftime("Arrival: %A %H:%M") )

Arrival: Sunday 21:31

Arrival: Friday 18:31

**13/11/2024 04:08 p.m. 1.449**

**Programa 97)**

**functions-v6.py**

def arrival\_time(destination, hours=51):

    now = datetime.now()

    arrival = now + timedelta(hours=hours)

    return arrival.strftime(f"{destination} Arrival: %A %H:%M")

"""

arrival\_time()

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

TypeError: arrival\_time() missing 1 required positional argument: 'destination'

"""

# The function now requires a destination argument.

# The destination argument is a required positional argument.

# The hours argument is a keyword argument with a default value of 51.

# The function calculates the arrival time based on the current time and the hours argument.

# The function returns the arrival time in the specified format.

# The function call without the destination argument raises a TypeError.

# The function call with the destination argument returns the arrival time.

arrival\_time("Moon")

print(arrival\_time("Moon"))

# Moon Arrival: Monday 06:45 (example output)

arrival\_time("Orbit", hours=0.13)

print(arrival\_time("Orbit", hours=0.13))

- Ejemplo similar al anterior. Ver los comentarios dentro del programa.

Moon Arrival: Sunday 21:37

Orbit Arrival: Friday 18:45

**13/11/2024 04:08 p.m. 1.449**

**Programa 98)**

**functions-v6 - copia.py**

- Igual al programa v6.

**13/11/2024 04:32 p.m. 1.388**

**Programa 99)**

**functions-v7 - copia.py**

- Igual al programa v7.

**13/11/2024 04:32 p.m. 1.388**

**Programa 100)**

**functions-v7.py**

"""

Use variable arguments in Python

In Python, you can use any number of arguments and keyword arguments without declaring each one of them.

This ability is useful when a function might get an unknown number of inputs.

Variable arguments

Arguments in functions are required. But when you're using variable arguments, the function allows any number of arguments

(including 0) to be passed in. The syntax for using variable arguments is prefixing a single asterisk (\*) before the argument's name.

The following function prints the received arguments:

"""

def variable\_length(\*args):

    print(args)

variable\_length()

()

variable\_length("one", "two")

('one', 'two')

variable\_length(None)

(None,)

"""

A rocket ship goes through several steps before a launch. Depending on tasks or delays, these steps might take longer than planned.

Let's create a variable-length function that can calculate how many minutes until launch, given how much time each step is going to take:

"""

print(" \*\*\*\*\* ")

def sequence\_time(\*args):

    total\_minutes = sum(args)

    if total\_minutes < 60:

        return f"Total time to launch is {total\_minutes} minutes"

    else:

        return f"Total time to launch is {total\_minutes/60} hours"

sequence\_time(4, 14, 18)

print(sequence\_time(4, 14, 18))

sequence\_time(4, 14, 48)

print(sequence\_time(4, 14, 48))

- Ver comentarios. Uso de número variable de argumentos con \*args (por convención) o algo similar. En este ejemplo se usa una función con número variable

Y opcional de argumentos y un IF para evaluar condiciones que muestren resultados en HORAS o MINUTOS.

**13/11/2024 04:40 p.m. 1.072**

**functions-v8.py**

**13/11/2024 04:40 p.m. 1.072 functions-v8 - copia.py**

- Igual al programa v8.

**13/11/2024 04:45 p.m. 515 functions-v9 - copia.py**

- Igual al anterior programa.

**13/11/2024 04:45 p.m. 515 functions-v9.py**

**13/11/2024 05:24 p.m. 745 errors-v1.py**

**13/11/2024 05:27 p.m. 874 config.py**

**13/11/2024 05:37 p.m. 221 errors-v2.py**

**13/11/2024 05:43 p.m. 389 errors-v3.py**

**13/11/2024 05:49 p.m. 754 errors-v4.py**

**13/11/2024 05:52 p.m. 876 errors-v5.py**

**13/11/2024 06:12 p.m. 1.290 errors-v6.py**

**13/11/2024 06:59 p.m. 169 repl-v1.py**

**16/11/2024 05:52 p.m. 147 type-v1.py**

**16/11/2024 06:16 p.m. 570 slice lists-v1.py**

**16/11/2024 06:50 p.m. 212 while-v2.py**

**16/11/2024 07:02 p.m. 156 loops-v1.py**

**16/11/2024 08:29 p.m. 223 pip\_request-v2.py**

**16/11/2024 08:34 p.m. 572 API\_successful\_request-v1.py**

**16/11/2024 08:47 p.m. 842 API\_successful\_request-v2.py**

**16/11/2024 08:54 p.m. 463 API\_test-v3.py**

**16/11/2024 08:57 p.m. 589 API\_test-v4.py**

**16/11/2024 09:07 p.m. 725 API\_test-v5.py**

**16/11/2024 09:10 p.m. 1.366 API\_test-v6.py**

**16/11/2024 09:14 p.m. 756 API\_test-v7.py**

**17/11/2024 09:47 a.m. 657 pip\_install\_tests\_v1.py**

**17/11/2024 09:52 a.m. 324 pip\_request-v1.py**

**03/01/2025 12:59 p.m. 226 prueba\_shortcut\_control\_D.py**

**03/01/2025 01:00 p.m. 250 prueba\_shortcut\_control\_D.\_mas\_control\_KCpy.py**

**03/01/2025 02:53 p.m. 76 Untitled-1.py**

**03/01/2025 02:57 p.m. 76 Untitled.py**