**Introducción al cómputo:**

* Al principio el computo se utilizaba para predecir la posición del sol y de los astros, para poder predecir las estacione y preparar cosas como la tierra y las batallas.
* El telar de Jacquard creaba tarjetas que representaban información. Nos dimos cuenta de que podemos separar los resultados de la información.
* Motor analítico de Babbage, una maquina general que pudiera realizar cálculos.
* En el siglo 19 los Estados Unidos no podían hacer censos bien porque se habían expandido mucho, así que llego una máquina para ello
* Antes una computadora era una persona cuya profesión es seguir instrucciones detalladas para llegar a un resultado, pero los cálculos tenían muchos errores.
* Turing y Charles encontraron que todos los algoritmos podían reducirse a una maquina con una cinta infinita donde estaban apuntados los símbolos. Se dieron cuanta que matemáticamente todos los algoritmos eran en principio el mismo.
* ENIAC fue la primera computadora digital, para programarla había que conectar y desconectar cables, el problema era que insertar un nuevo programa tomaba mucho tiempo, además utilizaba sistema decimal.
* Diagrama

  Descripción generada automáticamenteVon Neuman nos da la arquitectura de Von Neuman, él se da cuenta que dentro de los componentes electrónicos se pueden utilizar este hardware, así también se podía almacenar el programa, el genera la primera computadora a la cual se le puede introducir el programa y que se ejecutara ahí mismo
* Microchips fue lo que nos permitió llegar al computo actual. Hoy en día los microchips se construyen con luz y solamente se pueden ver con microscopios de electrones.
* La nube son solamente Datacenter, pero aún existen problemas que aun no se pueden resolver de manera computacional, por ejemplo, el clima.
* Richard Feynman nos dio las bases del computo cuántico, el pensaba que no podíamos simular sistemas cuenticos con una computadora cuántica.
* Las computadoras cuánticas operan a temperaturas cercanas al 0 absoluto.

Las computadoras hacen solamente dos cosas, cálculos y recordar el resultado de esos cálculos. Hoy en día se pueden realizar millones de cálculos e incluso así hay problemas que no sabemos como resolver y que existen problemas que nunca van a ser resueltos con computo.

**Introducción a los lenguajes de programación:**

Existe conocimiento declarativo, que nos dice que relaciones existe entre diversos objetos con una formula matemática, y conocimiento imperativo nos dice como llegar a un resultado, los algoritmos hacen parte de este último.

Un algoritmo es una lista finita de instrucciones que describen un cómputo, que cuando se ejecuta con ciertas entradas (inputs) ejecuta los pasos intermedios para llegar a un resultado (output), de esta idea de algoritmos nacieron los primeros lenguajes. Los lenguajes de programación comenzaron con Ada Lovelace, quien se dio cuenta que podía calcular una serie llamada la serie de los números de Bernoulli.

Esta forma de describir computo es una forma humana, pero una computadora entiende números binarios, programar con números binarios es muy difícil así que se desarrollaron los lenguajes de alto nivel, Grace Murray Hopper se da cuenta que puede escribir en binario una serie de instrucciones que leen otro programa y lo pueden traducir a un leguaje que pueda entender la computadora, este pensamiento nos dio los lenguajes de programación modernos.

Dennis Ritchie fue el creador del lenguaje C, y es muy importante porque es muy cercano al hardware y su sintaxis influyo en casi todos los lenguajes posteriores, C es como el latín de los lenguajes de programación. Guido van Rossum se dio cuenta que los lenguajes tienen que ser comprensibles, así que elimino casi todos los símbolos que nos hacían sentir que lo que estábamos leyendo era raro, por eso Python es tan fácil de leer.

Los lenguajes de programación modernos implementan todos los primitivos que en principio necesitamos para realizar cualquier tipo de algoritmo, se llaman Turing Completeness, además nos dan ciertas herramientas que hacen que escribir programas sea muy sencillo. Todos los lenguajes tienen una sintaxis (secuencia de símbolos que es correcta), una semántica estática (cual es la interpolación de símbolos significan algo) y una semántica (define cual de todos los posibles significados es el correcto), en los lenguajes de programación solo se preocupa de la sintaxis y de la semántica estática, porque todos tienen un solo significado, son preciso y exactos.

**Elementos básicos de Python:**

* Los lenguajes de alto nivel significan que están diseñados para los humanos, se acercan mucho mas al lenguaje natura, y los lenguajes de bajo nivel están optimizados para que una maquina pueda entenderlo
* Los lenguajes generales significan todos los primitivos que nos dijo Turing para poder implementar y computar cualquier algoritmo, y los lenguajes de dominio son lenguajes especializados que están tarjeteados a aplicaciones muy especificas
* Los lenguajes interpretados quieren decir que mientras corre el programa después de cada instrucción, esta se traduce a lenguaje máquina, y un lenguaje compilado quiere decir que nosotros convertimos antes de dárselo a la computadora el lenguaje en un lenguaje maquina

Texto

Descripción generada automáticamentePython es de alto nivel, es general, y es interpretado.

Literales: formas de inicializar objetos en memoria de manera que se anotan directamente en código.

Objetos: la forma en como se modela el mundo dentro del programa. Son valores en memoria que se pueden referencial con algún tipo de variable. Tienen tipos que pueden ser escalares y no escalares, lo que quiere decir si se puede o no dividir. (un vector se puede dividir, pero un entero no)

Interfaz de usuario gráfica

Descripción generada automáticamente

**Asignación de variables:**

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza bajaLas variables solo son el nombre que se vinculan con un valor de memoria, y se vincula a través de el operador de asignación. En programación un solo igual significa asignación, y se le puede asignar a un valor o a una expresión. Si le damos el nombre como a, x y o z nadie va a entender que significa la variable, lo ideal es dar nombres exactos de lo que significa la variable, o sea que deben tener un nombre que signifique algo para los humanos.

Gráfico

Descripción generada automáticamenteTexto

Descripción generada automáticamenteEs posible reasignar una variable, y simplemente se esta cambiando a donde apunta en memoria. El nombre seguirá siendo el mismo, cuando no hay ningún apuntador a ese lugar en especifico en memoria, quiere decir que el ‘garbage colector’ libera ese espacio en la memoria, hay lenguajes donde esto se tiene que hacer explícitamente

En Python las variables pueden tener mayúsculas, minúsculas, números (pero sin comenzar) y el guion bajo, además no pueden llamarse con palabras reservadas. Cada lenguaje tiene palabras reservadas en Python son:

Tabla

Descripción generada automáticamente

**Cadenas y entradas:**

Las cadenas son secuencias de caracteres, cuando se traban diferentes tipos, también cambiara el significado de los operadores. El (\*) lo que va a hacer es repetir la cadena por el numero de veces que se le indique, el (+) va a concatenar la cadena, Python permite concatenar de manera muy legible, para eso se utiliza la cadena de formato que se identificara con la letra f Al principio de la cadena:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza mediaA las cadenas también se le puede aplicar diferentes tipos de funciones, la función ‘len’ nos da la longitud de la cadena, también se puede acceder a cada uno de los elementos de esta cadena a través de índices, y también es posible dividir la cadena en rebanadas para obtener sub-cadenas.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteUn conjunto de letras blancas en un fondo blanco

Descripción generada automáticamente con confianza media

Las cadenas son inmutables, una ves se declaran en memoria ya no pueden modificarse, esto quiere decir que, si concatenamos la cadena inicial, no quiere decir que estemos cambiando la cadena, sino que se le esta asignado un espacio en memoria a la cadena ya concatenada y se le está apuntando.

Para los inputs o entradas se utiliza la función ‘input’ para recibir datos del usuario e ingresarlos al programa, pero esta función solo devolverá Strings. La función ‘print’ puede recibir múltiples argumentos.

Texto

Descripción generada automáticamente

Con la función ‘type’ se puede saber el tipo de variable, por tanto, si se le ingresa como parámetro una cadena String regresara un str, si queremos convertir un String a un entero se puede hacer de la siguiente manera:

Texto

Descripción generada automáticamente

**Programas ramificados:**

Las instrucciones deben poder tomar decisiones, o sea ver si se ejecuta una u otra parte del programa, para eso se utilizan ramificaciones. Para poder hacer ramificaciones se utilizan operadores de comparación, estos se utilizan para comprar dos variables.

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza bajaAdemás, también existen operadores lógicos:

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamenteLa forma más común para hacer programas ramificados es con un if-else, donde el valor de la condición (falso-verdadero) que tenga el if determinara si se ejecutan las expresiones de if o de else. En Python se utiliza indentacion para denotar la forma en como diferenciamos la condición del cuerpo de esta condición, en otros lenguajes se utilizan llaves.

La última forma para general programas ramificados es con elif, puede haber tantos como se necesiten.

**Iteraciones:**

También llamados loops, se utilizan cuando queremos que un programa haga lo mismo varias veces, se puede salir de una iteración con un ‘break’ y hay que tener cuidado con un loop infinito.

**Bucles for:**

Los bucles, en diversos lenguajes de programación pueden ser definidos o indefinidos. Los bucles definidos preestablecen las condiciones de la iteración por adelantado. Por su parte, los bucles indefinidos establecen la condición en la que una iteración terminará. En este último tipo de bucles existe el riesgo de que el bucle se vuelva infinito (cuando la condición de suspensión nunca se cumple). Los bucles definidos se implementan en Python a través del keyword for. Por su parte, los bucles indefinidos se implementan con el keyword while.

Una segunda forma de crear un bucle definido es iterando en una colección de objetos. Esta es la forma que Python utiliza. En la definición anterior debemos entender <iterable> como una Texto

Descripción generada automáticamentecolección de objetos; y la <variable> como el elemento específico que se está exponiendo mediante el bucle en cada iteración.

Texto

Descripción generada automáticamenteEn Python, un iterable es un objeto que se puede utilizar en un bucle definido. Si un objeto es iterable significa que se puede pasar como argumento a la función ‘iter’. El ‘iterable’ que se pasa como parámetro a la función ‘iter’ regresa un ‘iterator’. Todas las llamadas anteriores regresan un objeto de tipo ‘iterator’.

¿Qué pasa si le pasamos a la función iter un objeto que no es iterable? Obtendremos un TypeError que señala que el objeto no es un iterable. Esto es un ejemplo de programación defensiva en el que Python verifica el tipo del objeto antes de proceder al cómputo.

Es importante señalar que estos no son los únicos tipos de objetos que pueden ser iterable. Existen gran cantidad de ejemplos en la librería estándar y, de hecho, casi cualquier objeto se puede convertir en un iterable (pero eso ya lo veremos cuando hablemos de Python avanzado).

Un iterator es un objeto que regresa sucesivamente los valores asociados con el iterable.

Texto

Descripción generada automáticamente

el iterator guarda el estado interno de la iteración, de tal manera que cada llamada sucesiva a next regresa el siguiente elemento. ¿Qué pasa una vez que ya no existan más elementos en el iterable? La llamada a next arrojará un error de tipo StopIteration.

Para iterar a lo largo de un diccionario tenemos varias opciones:

* Ejecutar el bucle for directamente en el diccionario, lo cual nos permite iterar a lo largo de las llaves del diccionario.
* Ejecutar el bucle for en la llamada keys del diccionario, lo cual nos permite iterar a lo largo de las llaves del diccionario.
* Ejecutar el bucle for en la llamada values del diccionario, lo cual nos permite iterar a lo largo de los valores del diccionario.
* Ejecutar el bucle for en la llamada items del diccionario, lo cual nos permite iterar en una tupla de las llaves y los valores del diccionario

Texto

Descripción generada automáticamente

Podemos modificar el comportamiento de un bucle for mediante los keywords break y continue.

* break termina el bucle y permite continuar con el resto del flujo de nuestro  
  programa.
* continue termina la iteración en curso y continua con el siguiente ciclo de  
  iteración.

Python implementa los bucles definidos mediante los bucles for. Esta implementación nos permite iterar a lo largo de cualquier objeto que sea iterable. Para iterar necesitamos un iterador que nos regresará el siguiente valor en cada iteración. Todo esto, Python lo puede hacer por nosotros con el constructo for ... in ....

**Representación de flotantes**

¿Cómo representaremos 1/10 (que escribimos en Python cómo 0.1)? Lo mejor que podemos hacer con cuatro dígitos significativos es (0011, -101) que es equivalente a 3/32 (0.09375). ¿Qué tal si tuviéramos cinco dígitos significativos? La mejor representación sería (11001, -1000) que es equivalente a 25/256 (0.09765625). ¿Cuántos dígitos significativos necesitamos entonces? Un número infinito. No existe ningún número que cumpla con la siguiente ecuación: sim \* 2\*\*-exp.

En la mayoría de las implementaciones de Python tenemos 53 bits de precisión para números flotantes. Así que los dígitos significativos para representar el número 0.1 es igual a:

11001100110011001100110011001100110011001100110011001 que es equivalente al número decimal: 0.1000000000000000055511151231257827021181583404541015625

Muy cercano a 1/10 pero no exactamente 1/10. Ahora ya sabemos la razón de esa respuesta tan extraña. Hay muy pocas situaciones en la que 1.0 es aceptable, pero 0.9999999999999999 no. Pero ¿Cuál es la moraleja de esta historia?

Hasta ahora hemos verificado igualdad con el operador ==. Sin embargo, cuando estamos trabajando con flotantes es mejor asegurarnos que los números sean aproximados en vez de idénticos. Por ejemplo, x < 1.0 and x > 0.99999.

**Enumeración exhaustiva:**

Este algoritmo también se llama adivina y verifica, donde se enumeran todas las posibilidades hasta encontrar la respuesta, aunque sean muchas posibilidades esto no supone un problema porque los computadores actuales son lo suficientemente veloces, es una de las primeras opciones para resolver un problema y es de los más importantes.

**Aproximación de soluciones:**

Es similar a la enumeración exhaustiva, con la diferencia de que no necesita una solución exacta, aquí hay que definir qué tan cerca se quiere estar de la solución, y a la diferencia entre la realidad y la aproximación se va a llamar épsilon.

Hay que hacer un canje, no se puede ser preciso y rápido a la vez, mientras más preciso sea épsilon, más ciclos de cómputo son necesarios y más demorado será, pero si no se quiere ser tan exacto, se hace en menos tiempo.

Los algoritmos dependen del hardware, por eso este algoritmo es tan lento.

Tabla, Excel

Descripción generada automáticamente**Búsqueda Binaria:**

Es uno de los algoritmos mas eficiente, uno de los requisitos es que el conjunto de búsqueda debe estar ordenado, se puede utilizar en una raíz cuadrada, por ejemplo. En lo anteriores algoritmos había que probar todos los valores disponibles, pero en la búsqueda binaria es posible acortar el espacio de búsqueda a la mitad cada vez, y el espacio se va haciendo pequeño muy rápido. Supongamos que queremos encontrar el número 7 en una serie de números del 1 al 10:

Primero se evaluó si 5 era mayor que 7, y se acorta a la mitad el espacio de búsqueda, se hace el mismo procedimiento hasta encontrar el número 7.

**Funciones y abstracción**:

Texto

Descripción generada automáticamente con confianza mediaEs posible modularizar el código. La abstracción significa que no necesitas entender la forma en la que algo opera internamente para poderlo utilizar, por ejemplo, una calculadora que lo importante es saber operarla. Los ingenieros de software deben poder generar esas abstracciones, porque se utilizan librerías escritas por otras personas donde lo que interesa es como utilizarlo,

La decomposicion permite dividir el código en componentes, como pequeños programas dentro de un programa mayor que colaboran en conjunto para llegar a una solución. Encapsulamos la implementación del código, y se define una interfaz publica y generar muchos pedazos de código.

Pera definir una función se utiliza ‘def’ seguido del nombre, los parámetros luego el cuerpo de la función y finalmente un ‘return statement’, si no se coloca lo último, la función devolverá por defecto el valor ‘none’, que es la ausencia de valor.

keyword y valores por defecto: Las funciones en Python pueden tener argumentos de tipo keyword y valores por defecto, esto quiere decir que

Texto

Descripción generada automáticamente con confianza media

Esta función se puede llamar con los argumentos posicionales, simplemente se ve la posición de los argumentos y se llaman, si hay un valor por defecto se puede omitir, por ejemplo, el valor por defecto de inverso es ‘False’, pero también se puede llamar esta función con keywords, de esta manera se puede invertir el orden (como en la última línea). Una misma función puede ejecutarse de diversas formas.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**Scope o Alcance:**

Es importante entender el alcance de las funciones, cuando se ejecuta una función Python entra en un nuevo contexto de ejecución, y a este contexto se le asignan ciertas variables de las cuales se tienen acceso en el código. Aunque se hallan declarado variables antes, si no están en el contexto de ejecución, no se puede acceder a ellas.

También se pueden pasar funciones como argumentos, no solamente variables primitivas. El programa en principio se lee de arriba abajo y de izquierda a derecha, por lo que lo primero que hay son las funciones y luego la ejecución de estas, cuando se ejecutan las funciones se entra a otro contexto de ejecución y empieza a ve lo que hay dentro de la primera función donde a su vez se entra en un tercer contexto de ejecución (en func2).

Los contextos de ejecución van apareciendo o desapareciendo conforme se ejecutan funciones, hasta regresar al contexto global.

**Especificaciones del código:**

Hay que decirle al usuario como utilizar la función, en Python se llama docspring, eso se puede hacer con triple doble comilla, lo primero es que hace la función, después que significan los parámetros y cual es el tipo del parámetro y que se espera, en este caso estamos esperando dos enteros, y también hay que decir que regresa la función.

Texto

Descripción generada automáticamente

Para poder saber cómo funciona la función, simplemente con el comando help, y el nombre de la función entre paréntesis, para salir solamente se le da q. así se documentas las funciones y se pueden leer funciones que fueron hechas por otras personas.

buenas prácticas de documentación: <https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring>

**Recursividad:**

Texto

Descripción generada automáticamenteSe puede definir de manera algorítmica, como una forma de crear solución con el principio de divides y vencerás, o sea que un problema podemos resolverlo utilizando versiones mas pequeñas del mismo problema. También se puede definir de forma programática, es la recursividad, una técnica mediante la cual una función se llama a sí misma, esto se logra haciendo que cuando una función se ejecute, se llame a sí misma.

Las matemáticas también son una forma de representar cómputos. Para entenderlo es muy útil utilizar factoriales, que puede resolverse con un for loop, pero también con una función recursiva, la forma matemática de escribir la recursividad es:

Dibujo en blanco y negro

Descripción generada automáticamente con confianza baja

Una recursión es simplemente una forma de iterar, y cuando una función es recursiva, podemos representarla como un loop. En los factoriales el caso base es

En Python siempre se empieza el código escribiendo cual es el caso base, porque si no se puede entrar en un infinite loop. Python no permite llegar a un infinite loop pero si generara un error porque existe un límite en la recursividad, para saber el límite de recursividad de Python se importa la librería sys, y el límite se puede modificar, de la siguiente manera:

>>> import sys

>>> print(sys.getrecursionlimit())

1000

sys.setrecursionlimit(n) # n es el nuevo límite a establecer

**Fibonnacci y la Recursividad:**

La secuencia de Fibonacci es una función matemática que se define recursivamente. En el año 1202, el matemático italiano Leonardo de Pisa, también conocido como Fibonacci, encontró una fórmula para cuantificar el crecimiento que ciertas poblaciones experimentan.

Imagina que una pareja de conejos nace, un macho y una hembra, y luego son liberados. Imagina, también, que los conejos se pueden reproducir hasta la edad de un mes y que tienen un periodo de gestación también de un mes. Por último imagina que estos conejos nunca mueren y que la hembra siempre es capaz de producir una nueva pareja (un macho y una hembra). ¿Cuántos conejos existirán al final de seis meses?

Una forma de visualizar este crecimiento es mirándolo de forma tabular:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Mes | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| hembras | 1 | 1 | 2 | 3 | 5 | 8 | 13 |

Un punto importante a considerar es que para el mes n > 1, hembras(n) = hembras(n - 1) + hembras(n - 2).

Como podemos ver, tenemos una definición distinta a la de factorial que vimos anteriormente. En específico, tenemos dos casos base (0 y 1) y tenemos dos llamadas recursivas (hembras(n - 1) + hembras(n - 2)). Podemos crear una solución recursiva de manera sencilla:

def fibonacci(n):

if n == 0 or n == 1:

return 1

return fibonacci(n - 1) + fibonacci(n - 2)

**Funciones como objetos:**

Una de las características más poderosas de Python es que todo es un objeto, incluyendo las funciones. Las funciones en Python son “ciudadanos de primera clase”. Esto, en sentido amplio, significa que en Python las funciones:

* Tienen un tipo
* Se pueden pasar como argumentos de otras funciones
* Se pueden utilizar en expresiones
* Se pueden incluir en varias estructuras de datos (como listas, tuplas, diccionarios, etc.)

**Argumentos de otras funciones:**

Hasta ahora hemos visto que las funciones pueden recibir parámetros para realizar los cómputos que definen. Algunos de los tipos que hemos pasado son tipos simples como cadenas, números, listas, etc. Sin embargo, también pueden recibir funciones para crear abstracciones más poderosas. Veamos un ejemplo:

def multiplicar\_por\_dos(n):

return n \* 2

def sumar\_dos(n):

return n + 2

def aplicar\_operacion(f, numeros):

resultados = []

for numero in numeros:

resultado = f(numero)

resultados.append(resultado)

>>> nums = [1, 2, 3]

>>> aplicar\_operacion(multiplicar\_por\_dos, nums)

[2, 4, 6]

>>> aplicar\_operacion(sumar\_dos, nums)

[3, 4, 5]

**Funciones en expresiones:**

Una forma de definir una función en una expresión es utilizando el keyword lambda. lambda tiene la siguiente sintaxis: lambda <vars>: <expresion>.

Otro ejemplo interesante es que las funciones se pueden utilizar en una expresión directamente. Esto es posible ya que como lo hemos platicado con anterioridad, en Python las variables son simplemente nombres que apuntan a un objeto (en este caso a una función). Por ejemplo:

sumar = lambda x, y: x + y

>>> sumar(2, 3)

5

**Funciones en estructuras de datos:**

Las funciones también se pueden incluir en diversas estructuras que las permiten almacenar. Por ejemplo, una lista puede guardar diversas funciones a aplicar o un diccionario las puede almacenar como valores.

def aplicar\_operaciones(num):

operaciones = [abs, float]

resultado = []

for operacion in operaciones:

resultado.append(operacion(num))

return resultado

>>> aplicar\_operaciones(-2)

[2, -2.0]

Como pudimos ver, las funciones son objetos muy versátiles que nos permiten tratarlas de diversas maneras y que nos permiten añadir capas adicionales de abstracción a nuestro programa.

**Tuplas:**

Existen otro tipo de datos que se denominan datos estructurados, estos permiten agrupar muchos valores adentro de una sola variable. Las tuplas son secuencias inmutables de objetos, lo que quiere decir que es una lista de valores que no se pueden modificar, y los objetos son pedazos de memoria, entonces si tenemos una variable que apunta a una tupla no podemos modificarlo, se puede reasignar pero eso significa que estamos apuntando a otro objeto.

Las tuplas y las cadenas son secuencias inmutables, pero las tuplas pueden contener cualquier tipo de valor, y las cadenas solo Strings, además permite tener tuplas mixtas o sea con varios tipos de valores, además, las tuplas pueden ser utilizadas para devolver varios valores dentro de una función, eso es útil por ejemplo para calcular coordenadas.

Las tuplas se pueden definir simplemente con paréntesis, y se puede acceder a su contenido a través de indicadores

Texto

Descripción generada automáticamente

Cuado se tiene una tupla de un solo valor, la tupla va a tener el tipo de ese único valor, por eso se suele poner una coma después de ese valor, para que sea de tipo tuple

Texto

Descripción generada automáticamente

Es posible sumar las tuplas, y los que se hace es generar una nueva tupla y se está asignando.

Texto

Descripción generada automáticamente

Las tuplas es posible desempaquetarlas, como si quisiéramos asignar el dos a una x, el 3 a una y y el 4 a una z, en Python simplemente se definen varias variables y se asignan a la tupla.

Un conjunto de letras blancas en un fondo blanco

Descripción generada automáticamente con confianza media

Finalmente se puede regresar varios valores de una función:

Texto

Descripción generada automáticamente

**Rangos:**

Los rangos representan una secuencia de enteros, podemos definirlos con un comienzo, con un fin y con los pasos para las secuencias, también son inmutables al igual que las cadenas o las tuplas, esto hace que sean muy eficientes en gestión de memoria, porque desde el principio se sabe cuanta memoria va a ocupar. Los rangos se piensan normalmente dentro de los for loops:

Texto

Descripción generada automáticamente

En el caso anterior no se esta incluyendo el numero 5, porque los rangos siempre funcionan con estrictamente menor. Dos rangos pueden estar declarados de manera diferente pero tener l0os mismo valores, por eso en el siguiente ejemplo ambos rangos son iguales.

Texto

Descripción generada automáticamente



Si se observara la ubicación de los rangos anteriores, tendrían dos direcciones de memoria diferentes porque ambos objetos son diferentes.

* Object equality: es cuando se compara si dos objetos son el mismo objeto, se utiliza el operador (is)
* Value equality: es cuando se compara el valor de dos objetos, se utiliza el operador (==)

**Listas y mutabilidad:**

Las listas son secuencias de objetos que si son mutables, o sea que si podemos modificarlos, esto significa que se puede ir creciendo la memoria porque Python no sabe que tan grande va a ser la lista. Cuando se modifica una lista pueden existir efectos secundarios (side effects), eso quiere decir que cada ves que se muta una lista, es una posible fuente de bugs en el programa. También es posible iterar a lo largo de las listas para ver que objetos tiene, para esto se puede utilizar un for loop, además pueden contener cualquier tipo de valor.

Para modificar una lista se puede directamente cambiar el valor dentro de un índice (my\_list[0]=5) o utilizar los métodos de las listas: append, pop, remove, insert. Etc.

* Append = añade un valor al final de la lista
* Pop = elimina el ultimo valor de la lista

En las listas se puede utilizar la notación de slices (así como en los rangos) esto generará una nueva lista

Texto

Descripción generada automáticamente

Iteración con un for loop:

Texto

Descripción generada automáticamente

Cuando generamos una lista y luego la igualamos a otro valor, lo que realmente esta sucediendo es que ambos nombres de las variables están apuntando al mismo lugar, esto se puede verificar con la función id (id es para ver la posición en memoria)

Texto

Descripción generada automáticamente

Esto es importante tenerlo en cuenta porque puede generar errores, aquí a y b son de hecho el mismo objeto ya que tienen la misma dirección en memoria, por tanto si modificamos a, también se modificará b, y si creamos una nueva lista que contenga a y b, también se modificará.

Texto

Descripción generada automáticamente

Existen patrones dentro del desarrollo web que prohíben tener datos mutables y obligan a generar copias. Una forma de evitar estos problemas es clonando la lista, casi siempre es mejor clonar una lista que modificarla. Para eso se puede utilizar la notación de rebanadas que siempre generara una nueva lista, o se puede utilizar la función list que viene incluida en Python.

La funcion list generará dos objetos, y tendrá dos valores en memoria diferentes, entonces si posteriormente se modifican, no van a haber el problema que se presentó anteriormente.

Texto

Descripción generada automáticamente

Una forma mas compacta de clonar una lista puede ser con la notación de slices, entonces d tendrá su propia dirección en memoria:

Texto

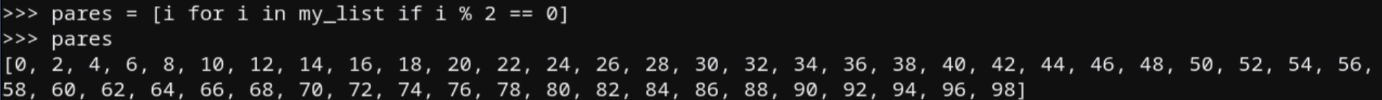
Descripción generada automáticamente

Los list comprehension es una forma concisa de aplicar operaciones a todos los valores de una lista o de una secuencia, también se pueden aplicar condiciones para definir a que valores se le aplicara la operación como si fuera un filtro.

Calendario

Descripción generada automáticamente

Si quisiéramos sacar únicamente los números pares lo haríamos de la siguiente manera:



Métodos de las listas:

* lista.extend(iterable) #extiende la lista con valores dentro de un iterable como un range()
* lista.insert(i, ‘valor’) #Agrega un valor en la posición i y recorre todos los demás. No borra nada.
* lista.pop(i) #Elimina valor en la posición i de la lista.
* lista.remove(‘valor’) #Elimina el primer elemento con ese valor.
* lista.clear() #Borra elementos en la lista.
* lista.index(‘valor’) #Retorna posición del primer elemento con el valor.
* lista.index(‘valor’, start, end) #Retorna posición del elemento con el valor dentro de los elementos desde posición start hasta posición end)
* lista.count(‘valor’) #Cuenta cuántas veces esta ese valor en la lista.
* lista.sort() #Ordena los elementos de mayor a menor.
* lista.sort(reverse = True) #Ordena los elementos de menor a mayor.
* lista.reverse() #Invierte los elementos
* lista.copy() #Genera una copia de la lista. También útil para clonar listas.

**Diccionarios:**

Los diccionarios son como listas, y se acceden a los valores a través de llaves en vez de índices numéricos. No tienen un orden interno, en otros lenguajes se llaman hash maps, por que la forma en como se establecen las llaves que se pasan a través de una función llamada hash, básicamente lo que generan los hash son valores únicos que permiten acceder de manera muy eficiente a los valores.

Qué es una función Hash? Una función hash es método para generar claves o llaves que representen de manera unívoca a un documento o conjunto de datos. Es una operación matemática que se realiza sobre este conjunto de datos de cualquier longitud, y su salida es una huella digital, de tamaño fijo e independiente de la dimensión del documento original. El contenido es ilegible.

Los diccionarios son mutables y pueden iterarse a través de sus llaves, a través de sus valores o a través de sus llaves y valores.

Para poder escribir diccionarios se utiliza la notación de llaves.

Texto

Descripción generada automáticamente

Si no sabemos si la llave existe o no, es posible utilizar el método get, donde nos devuelve un valor que definimos si no existe la llave, pero si efectivamente existe, devolverá el valor asignado a esa llave.

Texto

Descripción generada automáticamente

Y se puede asignar nuevos valores y agregar llaves

Imagen que contiene Texto

Descripción generada automáticamente

También es posible eliminar valores con la palabra clave del, simplemente se le da la llave que se quiere borrar:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para iterar es necesarios utilizar el método keys para imprimir las llaves, el método values para imprimir los valores, y para poder iterar por la llave y el valor agregando dos parámetros al for loop y con el método ítems

Texto

Descripción generada automáticamente

Existen ocasiones en las que se necesita saber si una llave esta dentro del diccionario, para eso se puede utilizar el operador in:

Texto

Descripción generada automáticamente

Python tiene cuatro tipos primitivos: enteros, flotantes, booleanos y cadenas o Strings.

**Dictionary Comprehension:**

La forma general para acceder a los valores de un diccionario a través de Dictionary Comprehension es:

Dict\_variable = {key:vakue for (key, value) in dictionary.items()}

Y el metodo se puede volver cada ves mas complejo a medida que se agregan condiciones:

Imagen que contiene Calendario

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En el ejemplo se esta creando un nuevo diccionario a partir de los valores de dict, y simplemente se están multiplicando los valores por dos, esto también es posible hacerlo con las llaves.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Estos métodos pueden ser usados para sustituir for loops y funciones lambdas, pero no todo puede convertirse.

**Pruebas de caja negra:**

Es necesario encontrar y resolver los bugs dentro del código, las pruebas de caja negra se basan en la especificación de una función o del propio programa, lo que hacen es probar que la funcion regrese ciertos outpus cuando le damos ciertos inputs, y tienen ese nombre porque se asume que no se conoce la implementación, simplemente se le dan inputs, y que la función regrese los outputs correctos, esto es importante para:

* Unit testing: son pruebas unitarias, y prueban función por función y se cercioran de que cada pedazo del código funciona
* Integration testing: es cuando se prueba si todos los módulos funcionan entre si

Si tenemos un background de electrónica, un unit test es probar que un led prende, y integration test es probar que todo el sistema funciona.

Unittest es el módulo en Python para realizar pruebas, para utilizarla se implementa como parámetro en una función (class CajaNegraTest(unittest.TestCase):) y dentro se deifnten test con la palabra def, y utilizando self en el resultado.

* Test driven development: antes de escribir la función, se escribe las pruebas que se esperan de la función.

Cuando queremos plantear las pruebas, hay que plantearse los casos posibles, en una función de suma pueden sumarse dos positivos, dos negativos, mixto o que el resultado de 0, esto se realiza para detectar los posibles errores de la función. En Python se utiliza la palabra clave self dentro de la función de test.

Una ves ya se tiene un test funcionando, y se siguen agregando test, se tiene una garantía de que si se modifica la función y el comportamiento cambia, el test va a proteger.

**Pruebas de caja de cristal:**

Son otra técnica para hacer pruebas, se basan en el flujo del programa, o sea que asumen que sabemos la implementación, y que podemos probar todos los posibles caminos que existen dentro de la función: ramificaciones, bucles for y while, recursión. Etc.

Estas pruebas son muy útiles en regression testing o mocks, esto quiere decir que se descubren los bugs cuando el programa ya salió a la luz, y hay que determinad donde esta ese bug.

Con un if hay que probar todas las conducciones, con un loop hay que probar un test donde no se entre al loop, otro donde se entre una vez y otro donde se entre más de una vez. En un while se prueba un test cuando al condición de entrada es falsa, y otra para ver como se comportan los break statements. Finalmente es muy importante probar todas las excepciones posibles en el código, o sea todos los posibles errores.

En ejemplo es utilizando una función para evaluar si la edad es mayor de 18, donde hay dos caminos y por tanto hay que hacer dos test, uno para cuando es verdadero y otro para cuando es falso.

**Debugging:**

La mejor forma de evitar bugs es con test, pero no es suficiente.

Grace Hooper fue una de las pioneras del cómputo, también fue la primera persona en encontrar un bug en una computadora, el bug era un insecto dentro de la computadora, hoy en día se siguen teniendo bugs virtuales y para hallarlos hay que seguir las siguientes reglas:

* No molestarse con el bug y aprender a utilizar el print statement: el print statementda una mirada a lo que sucede dentro de los programas
* Estudiar los datos disponibles
* Utilizar los datos para crear hipótesis y experimentos: generar pruebas (test)
* Mantener una mente abierta: no te preguntas porque esta fallando sino porque computa de es amanera, entender el programa.
* Llevar un registro de bugs en forma de test

Debuggear un programa es una habilidad que se aprende (learn skill), y es importante ser sistemático, porque lo importante es generar un test para que ese bug no vuelva a suceder.

Diseño de experimentos: debuggear es un proceso de búsqueda y cada prueba debe acotar el espacio de búsqueda, para eso es utilizar print statement o utilizando la búsqueda binaria hasta encontrar el pedazo de código donde falla la lógica

Errores comunes:

* Encontrar los sospechosos comunes: argumentos en un orden incorrecto, un nombre mal escrito, no inicializar una variable, tratar de comparar dos flotantes con igualdad exacta en lugar de una aproximación, utilizar == en lugar de is, efectos secundarios en las listas. Etc.
* Es posible que el bug no este donde se cree que esta, por eso es importante ser sistemático y utilizar búsqueda binaria
* Explicarle el problema a otra persona, una persona sin contexto
* Tener un registro de lo que se ha tratado de preferencia en test
* Dormir bien y no te frustres😉

**Manejo de excepciones:**

Las excepciones son una forma de manejar errores en el código, y son muy comunes en la programación, se pueden utilizar por ejemplo cuando sabemos que se va a dividir algo entre 0, a esto se le llama defensing programing. La mayoría de las excepciones se relacionen con errores de semántica, como operaciones que no son compatibles, cuando no se maneja una excepción el programa explota. (termina en error)

Las excepciones en Python se manejan con las siguientes palabras clave: try, except, finally.

El manejo de excepciones se utiliza en todos los lenguajes, y también pueden ser utilizados para ramificar el programa, por ejemplo tratar algo y si esto no funciona se puede continuar con el programa dentro de la estructura de la excepción.

Una pésima practica de programación es silenciar las excepciones, esto significa que en lugar de manejar el error simplemente se imprime, y esto hace que el programa tenga una fuente de bugs.

Para generar una excepción se utiliza la palabra raise, simplemente se escribe raise y el nombre de la excepción y esto significa que alguien más que utilice la función, tendrá que manejar esa excepción.

Para trabajar con el código se puede poner el try-except fuera de la función, o sea si simplemente es usuario, o ponerlo dentro de la función haciendo programación defensiva.

Excepciones comunes:

* ImportError: una importación falla.
* IndexError: una lista se indexa con un número fuera de rango.
* NameError: se usa una variable desconocida.
* SyntaxError: el código no se puede analizar correctamente
* TypeError: se llama a una función en un valor de un tipo inapropiado.
* ValueError: se llama a una función en un valor del tipo correcto, pero con un valor inapropiado.
* ZeroDivisionError: división entre 0.

**Excepciones como control de flujo:**

Con el uso de excepciones también es posible controlar el flujo del programa, la estructura de control de flujo que ofrece Python es (if…elif…else), entonces, ¿por qué es necesaria otra modalidad para controlar el flujo? Una razón muy específica: el principio EAFP (easier to ask for forgiveness than permission, es más fácil pedir perdón que permiso, por sus siglas en inglés).

El principio EAFP es un estilo de programación común en Python en el cual se asumen llaves, índices o atributos válidos y se captura la excepción si la suposición resulta ser falsa. Es importante resaltar que otros lenguajes de programación favorecen el principio LBYL (look before you leap, revisa antes de saltar) en el cual el código verifica de manera explícita las precondiciones antes de realizar llamadas, como JavaScript.

Texto

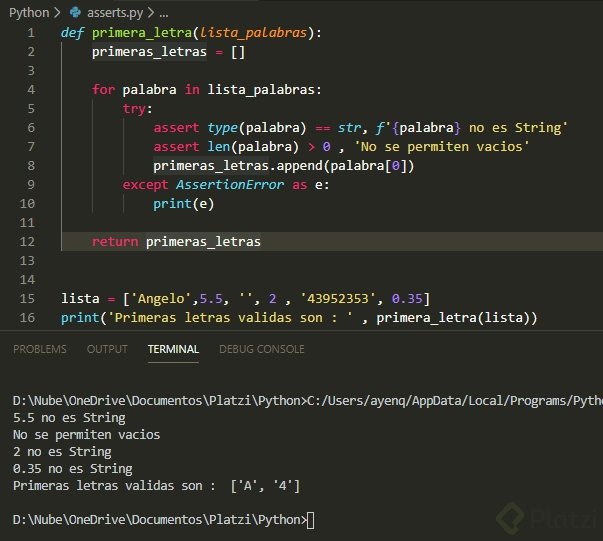
Descripción generada automáticamenteTexto

Descripción generada automáticamente

eE código de Python accede directamente a la llave y únicamente si dicho acceso falla, entonces se captura la excepción y se provee el código necesario. En el caso de JavaScript, se verifica primero que la llave exista en el objeto y únicamente con posterioridad se accede.

Es importante resaltar que ambos estilos pueden utilizarse en Python, pero el estilo EAFP es mucho más "pythonico".

**Afirmaciones:**

Las afirmaciones es un mecanismo para determinar si una condición se cumple o no, y seguir con la ejecución del programa o terminarlo. Es un método de programación defensiva, esto quiere decir que se prepara el programa para determinar si los inputs son del tipo que se espera, por ejemplo si se trabaja con un programa que solo trabaja con enteros, si el tipo no es correcto el programa puede fallar o dar un bug de lógica que va a ser muy difícil de encontrar.

Las afirmaciones también es in gran método para debuguear, por ejemplo cuando solo se quieren resultados y una ves en determinado punto de ejecución, se quiere terminar con el programa o con el loop.

Las afirmaciones se generan con la palabra assert, después dar una expresión booleana, y agregar un mensaje el que se tiene acceso a él una vez termina la ejecución si la assercion no funcion, si en caso contrario funciona, simplemente el programa sigue ejecutándose.

La siguiente función recibe una lista de palabras y toma la primera letra de cada palabra, y regresa una nueva lista con estas letras, en este caso probablemente se quiere trabajar con Strings, así que se crea un assert para eso, otra cosa para verificar es si el String contenga algún caracter, porque existen string vacíos.

Texto

Descripción generada automáticamente

En este caso se esta asegurando que los usuarios no cometan un error que puede ser fatal para la ejecución del programa.

**Conclusiones:**

Lo importante es el desarrollo de las herramientas cognitivas para la carrera de computer scientist