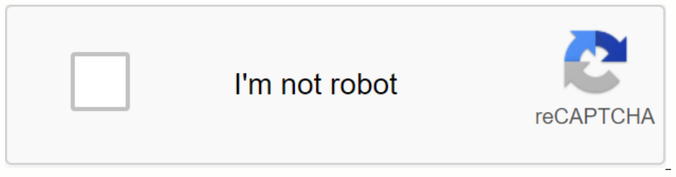
Certificado raton perez para imprimir pdf



Continue

Python es un lenguaje de programación orientado a objetos muy simple pero muy potente. La sintaxis de Python es muy simple para que un principiante pueda aprender Python con facilidad. He cubierto el lenguaje Python en muchos tutoriales de Python separados, esta es la página principal del tutorial de Python que tiene enlaces a todos los tutoriales que he compartido en Python. Este tutorial es tanto para principiantes como para estudiantes avanzados de Python. Te recomiendo que leas y practiques los tutoriales en el orden dado. Conceptos básicos de aprendizaje de Python: 1. Introducción a la programación de Python 2. Cómo instalar python en su sistema 3. Instalar PyCharm IDE en Mac OS X, Windows, Linux / Unix - Este es el IDE que usamos en todos los tutoriales. Puede escribir, compilar y ejecutar programas Python en este IDE sin esfuerzo. 4. Creación del primer proyecto de Python en PyCharm IDE 5. Declaraciones y bucles de control de comentarios de Python: 1. If Statement 2. Si.. otra declaración 3. Si.. Elif.. otra declaración 4. Inserte si. otra declaración 5. para el bucle 6. mientras que el bucle 7. declaración de parada 8. continuación de la declaración 9. Tipos de datos de instrucciones de éxito: 1. Números de Python 2. Lista de Python 3. Cadena de Python 4. Modos python de Python tupla Python: 1. Funciones de Python con el ejemplo 2. Ejemplo de Flashback a Python de OOPS: 1. Python OOPS 2. Clase de Python y objeto 3. Ejemplos de fabricantes en Python: pruebe estos ejemplos para practicar varios temas de programación de Python. Ejemplos de programación de Python Con el editor Pruébelo usted mismo, puede editar código Python y ver el resultado. print(Hi, World!) pruébalo tú mismo » Haga clic en probarlo usted mismo para ver cómo funciona. Manejo de archivos Python En la sección Administración de archivos, aprenderá a abrir, leer, escribir y eliminar archivos. Python file handling python database management en nuestra sección de base de datos aprenderá a acceder y trabajar con bases de datos mySQL y MongoDB: Python MySQL Tutorial Python MongoDB Tutorial Python Ejercicios Python Ejemplos de Python Aprender de Ejemplos! Este tutorial complementa todas las explicaciones con ejemplos clarificadores. Vea todos los ejemplos de Python Quiz Test de sus habilidades de Python con un cuestionario. Python Quiz Python Referencia También encontrará referencias completas de funciones y métodos: Referencia Visión General Funciones Incorporadas Métodos de Cadena Lista / Métodos de Matriz Diccionario Métodos Métodos Métodos Métodos Métodos Métodos Métodos Métodos Métodos Métodos Python Excepciones Python Pyth no Glosario Aleatorio Módulo Solicita Sección Módulo Matemático CMath Módulo CMath Descargar Python Descargar Python Desde el sitio web oficial de Python: Python Exams - Obtener su licencia! La solución perfecta para que necesitan equilibrar el trabajo, la familia y la construcción de carreras. ¡Ya se han emitido más de 25 000 certificados! Obtenga su certificado » El certificado HTML documenta su conocimiento de HTML. El certificado CSS documenta su conocimiento de CSS avanzado. El Certificado JavaScript documenta su conocimiento de javascript y HTML DOM. Python. El certificado jQuery documenta su conocimiento de jQuery. El certificado SQL documenta su conocimiento de SQL. El certificado PHP documenta su conocimiento de PHP y MySQL. El certificado XML documenta su conocimiento de XML, XML DOM y XSLT. El certificado Bootstrap documenta su conocimiento del marco de trabajo de Bootstrap. Tutorial de Python para principiantes con ejemplos. Aprenda Python en un día siguiendo esta entrada de blog. Comencemos a aprender la programación de Python en un día para comenzar con el lenguaje Python. Todo lo mejor para tu futuro y feliz aprendizaje de pitones. Tutorial de Python para principiantes La primera pregunta, ¿qué es Python? Según Guido van Rossum, el padre de Python, Python es: Un lenguaje de programación de alto nivel cuya filosofía básica de diseño es la legibilidad del código y la sintaxis que permite a los desarrolladores expresar sus ideas en muy poco código. Para mí, la razón principal para aprender Python es que es un lenguaje que se puede programar con gracia. Es fácil y natural escribir código e implementar mis ideas. ¿Para qué se utilizan los Python? Otra razón es que podemos usar python en muchos lugares: ciencia de datos, desarrollo web, aprendizaje automático, etc., todo se puede desarrollar usando Python. Google, Quora, Pinterest y Spotify utilizan Python para desarrollar web de fondo. Vamos a averiguar sobre Python ahora. Conceptos básicos de Python para principiantes Aquí vamos con el aprendizaje de Python en un día. 1. Variables de Python Puede pensar en una variable como una palabra para guardar un valor. Veamos un ejemplo. Es fácil establecer una variable en Python y asignar un valor. Si quieres guardar el número uno en la variable uno, vamos a probarlo: uno 1 Súper fácil? Sólo tiene que asignar un valor de 1 a la variable uno dos a 2 some_number a 10000 Durante el tiempo que desee, puede asignar cualquier valor a cualquier otra variable. Como puede ver en lo anterior, la variable dos almacena la variable entera 2 y la variable some_number guarda 10000. Además de un entero, también podemos usar True/False, String, Float y otros tipos de datos. • booleanstrue_boolean de Truefalse_boolean á Falso stringmy_name de Leandro Tk floatbook_price 15,80 2. Flujo de control de Python: la instrucción condicional si utiliza una expresión para determinar si una instrucción es true o false. Si es true, ejecute el código if, como se muestra en el siguiente ejemplo: if true: print(Hello Python 2 > 1: print(2 es mayor que 1) 2 es mayor que 1, por lo que se ejecuta el código de impresión. Cuando la expresión sobre si es false, se ejecutará la otra instrucción. si 1 > 2: print(1 es mayor que 2) else: print(1 no es mayor que 2) 1 es menor que 2, por lo que el código en el otro se ejecutará. También puede la declaración elif: si 1 > 2: printing(1 es mayor que 2)elif 2 > 1: printing(1 is not mayor than 2)else: printing(1 es igual a 2) 3. Bucle de Python y repetición en Python, podemos repetir de diferentes maneras. Voy a hablar un rato sobre. Bucle de Python While: mientras que la frase es true, mientras que el bloque de código dentro se ejecutará. Por lo tanto, el siguiente código imprime 1 a 10. Num á 1while num <-10: print(num) num + - 1 El bucle cuando se necesita una condición de bucle, y si la condición, la condición siempre es verdadera, se repite todo el tiempo, con una condición de bucle falso cuando el valor de num es 11. Otro fragmento de código puede ayudarle a comprender mejor el uso de la instrucción mientras: loop_condition - Truewhile loop_condition: print(Loop Condition maintain: %s %(loop_condition)) loop_condition - False La condición de bucle es True para que se repita hasta que sea falso. Python For Loop : Puede aplicar la variable num al bloque de código, y la sugerencia para se repetirá encima de él por usted. Este código imprimirá el mismo código que: 1 a 10. para i en el área (1, 11): print(i) ¿Viste? Eso es muy simple. El rango de i comienza de 1 a la undécima parte (10 es el décimo elemento). Lista de Python: colección ? Tabla ? estructura de datos Supongamos que desea guardar el entero 1 en una variable, pero también desea guardar 2 y 3, 4, 5 ... En lugar de usar cientos o miles de variables, ¿tengo otras formas de almacenar estos enteros que quiero almacenar? Como ya has adivinado, hay otras maneras de salvarlos. El directorio es una colección que puede almacenar una lista de valores (al igual que lo que desea guardar), a continuación, vamos a usarlo: my_integers [1, 2, 3, 4, 5] Esto es muy fácil. Creamos una tabla llamada my_integer y pusimos los datos en ella. Tal vez usted puede preguntar, ¿Cómo puedo obtener el precio en la tabla? Pregúntale a las buenas noticias. La lista tiene un concepto llamado indexación. La siguiente tabla del primer elemento es el índice 0 (0). El segundo indicador es 1, y así sucesivamente, usted necesita entender. En la sintaxis de Python, también es bueno entender: my_integers de impresión de 5, 7, 1, 3, 4] (my_integers[0]) - 5print(my_integers[1]) - 7print(my_integers[4]) #4 Si no desea guardar el número completo. Sólo quieres guardar algunas cadenas, como la lista de nombres de tus parientes. Mi aspecto es similar a este: relatives_names [Toshiaki, Juliana, Yuji, Bruno, Kaio] impresión (relatives_names[4]) #Kaio Principle es lo mismo que almacenar un entero, muy amigable. Solo aprendimos cómo funciona un puntero de lista, y también necesito decirle cómo agregar un elemento a la estructura de datos de la lista un elemento de la lista). La forma más común de agregar nuevos datos a la lista es la soldadura. Echemos un vistazo a cómo se utiliza: estante - [] bookhelf.append(El ingeniero eficaz) ingeniero) Impresión de 4 horas de trabajo (estante[0]) - La Engineerprint efectiva (estantería[1]) - El trabajo de 4 horas W Costura súper fácil, sólo necesita poner un artículo (como una máquina válida) como los parámetros de soldadura. Bueno, la lista de conocimiento es suficiente, echemos un vistazo a las otras estructuras de datos. Python Dictionary: Key-value data structure Now we know that the list is an indexed integer set. Pero, ¿qué pasa si no usamos enteros como índices? Podemos usar otras estructuras de datos, como números, cadenas u otros tipos de índices. Vamos a averiguar acerca de esta estructura de datos del diccionario. Un diccionario es una colección de pares clave-valor. El diccionario es tan largo: dictionary_example de la clave1: value1, key2: value2, key3: value3 - Key is the index in . ¿Cómo podemos acceder al precio del diccionario? Tienes que adivinar, ese es el uso de la llave. Vamos a probar dictionary_tk nombre: Leandro, apodo: Tk, nacionalidad: Brasileño - print(Mi nombre es %s %(dictionary_tk[nombre])) - Mi nombre es Leandro print(Pero se me puede llamar %s %(dictionary_tk[apodo])) - Pero se me puede llamar Tk print(Y por la forma en que soy %s %(dictionary_tk[nacionalidad])) - Y por la forma en que soy brasileño Tenemos un valor clave (edad) (24) que utiliza una cadena como la clave de valor entero . Creé un diccionario para mí que contiene mi nombre, apodo y nacionalidad. Estos atributos son las claves del diccionario. Del igual que hemos aprendido al usar un índice para tener acceso a una lista, también usamos un índice (la clave del diccionario es un índice) para tener acceso al valor almacenado en el diccionario. A medida que usamos la lista, vamos a aprender a agregar elementos al diccionario. El diccionario muestra principalmente el valor de la clave. Lo mismo se aplica cuando añadimos datos: dictionary_tk nombre: Leandro, apodo: Tk, nacionalidad: Brasileño, edad: 24o de impresión (Mi nombre es %(dictionary_tk[nombre])) - Mi nombre es Leandro Print(Pero me puede llamar %s %(dictionary_tk[nickname]dictionary_tk) dictionary_tk [nacionalidad])) - Y por cierto soy brasileño sólo tengo que señalar una clave en un diccionario en un vvalue. No hay dificultades, ¿verdad? Repetición de Python: Recorrer la estructura de datos Como aprendimos en los conceptos básicos de Python, la lista de repetición es simple. Los desarrolladores de Python solemos usar para bucles. Vamos a probarlo estante - [El ingeniero eficaz, La semana de trabajo 4 horas, Cero a uno, Inicio magro, Enganchado] para el libro en el estante: (libro) Extremadamente fácil y y es la belleza de Python. También podemos usar el bucle para la estructura de datos hash, pero debemos usar la clave: diccionario - some_key: some_value á para la clave en el diccionario: print(%s -> %s %(key, dictionary[key])) ? some_key –> some_value Lo anterior es un ejemplo de cómo utilizar un bucle para un diccionario. Para cada clave del diccionario, imprimimos el valor de la clave y la clave. Diccionario á some_key: some_value para la clave, valor en dictionary.items(): print(%s -> %s %(key, value)) - some_key –> some_value Hemos nombrado la clave y el valor de los dos parámetros, pero esto no es necesario. Somos libres de llamarlo. Echemos un vistazo: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 dictionary_tk nombre: Leandro, apodo: Tk, nacionalidad: Brasileña, edad: 24o para atributo, valor en dictionary_tk.items(): print(My %s es %s %(attribute, value)) - Mi nombre es Leandro - Mi apodo es Tk - Mi nacionalidad es brasileña - Mi edad es 24 Puede ver que usamos el atributo como un parámetro básico en el diccionario que lo define, que tiene el mismo efecto que el uso de una clave. Realmente genial! Clases y objetos de Python Algunas teorías: Los objetos son representaciones de entidades reales, como coches, perros o bicicletas. Estos objetos comparten dos características principales juntas: datos y comportamiento. Los coches tienen datos como el número de ruedas, el número de puertas y el espacio del asiento, y pueden mostrar su comportamiento: pueden acelerar, parar, mostrar cuánto combustible queda, y más. Tratamos los datos como características y comportamientos en la programación orientada a objetos. Una vez más expresado como: → de datos Propiedades y comportamientos → Clase de métodos es un plan para crear un único objeto. En el mundo real, a menudo encontramos muchos objetos del mismo tipo. Por ejemplo, coches. Todos los coches tienen la misma estructura y modelo (todos con motor, ruedas, puertas, etc.). Cada coche se fabrica con el mismo diseño y tiene los mismos componentes. Modelo de programación orientado a objetos de Python: ON Python, como lenguaje de programación orientado a objetos, tiene el significado de clases y objetos. Una clase es un diseño que es un modelo del objeto. Por lo tanto, una clase es un modelo o una forma de definir propiedades y comportamientos (como se describe en la sección teórica). Por ejemplo, una clase de vehículo tiene sus propias propiedades que determinan el tipo de vehículo que es ese objeto. Las características de un coche son el número de ruedas, el tipo de energía, la capacidad del asiento y su velocidad máxima. Con eso en mente, echemos un vistazo a en la sintaxis de las clases python: por encima del código, usamos la instrucción class para especificar una clase. ¿No es fácil? Un objeto es una creación de clase, que podemos unir por el nombre de clase. coche : Impresión del vehículo () (coche) <__main__. Instancia del vehículo, a la vertencia de 0x7fb1de6c2638,> Aquí, el coche es un objeto (o instancia) </__main__. Vehículo> </__main__. Vehículo> categoría de vehículo. Recuerde que la categoría de vehículos tiene cuatro características: el número de ruedas, el tipo de depósito de combustible, la capacidad del asiento y la velocidad máxima. Cuando creamos un nuevo objeto de vehículo para establecer todas las características. Por lo tanto, aquí, definimos una clase que acepta parámetros cuando se prepara: categoría de vehículo: __init__ def (self, number_of_wheels, type_of_tank, seating_capacity, maximum_velocity): self.number_of_wheels - number_of_wheels self.type_of_tank - type_of_tank self.seating_capacity - seating_capacity self.maximum_velocity - maximum_velocity Este método init. A eso lo llamamos fabricante. Así que cuando creamos un objeto de vehículo, podemos determinar estas propiedades. Imagina que nos gusta el Tesla Model S, así que queremos crear un objeto de este tipo. Tiene cuatro ruedas, utiliza electricidad, cinco asientos y una velocidad máxima de 250 kilómetros (155 millas). Comencemos creando un objeto como este: tesla_model_s - Vehículo (4, eléctrico, 5, 250) Cuatro + energía + cinco + velocidad máxima 250 km. Se establecen todas las propiedades. Pero, ¿cómo podemos acceder a estos valores de propiedad? Enviamos un mensaje al objeto para solicitarle este valor. A esto lo llamamos método. Es el comportamiento del objeto. Vamos a lograrlo: categoría de vehículo: __init__ def (auto, number_of_wheels, type_of_tank, seating_capacity, maximum_velocity): self.number_of_wheels - number_of_wheels self.type_of_tank - type_of_tank self.seating_capacity - seating_capacity self.maximum_velocity - maximum_velocity def number_of_wheels (auto): devolución self.number_of_wheels set_number_of_wheels def (solo, número): self.number_of_wheels número Esta es la aplicación de ambos métodos number_of_wheels y set_number_of_wheels. Lo llamamos getter & setter. Dado que la primera función es obtener el valor de propiedad, la segunda es establecer un nuevo valor para la propiedad. En Python, podemos usar @property (modificador) para determinar captadores y establecedores. Echemos un vistazo al código real: categoría de vehículo: def __init__ (number_of_wheels, type_of_tank, seating_capacity, maximum_velocity): self.number_of_wheels number_of_wheels self.type_of_tank - type_of_tank self.seating_capacity - seating_capacity self.maximum_velocity maximum_velocity @property number_of_wheels (auto): return self.number_of_wheels @number_of_wheels.setter def number_of_wheels (self, número): self.number_of_wheels número y podemos utilizar estos métodos como propiedades: tesla_model_s - Vehículo (4, eléctrico, 5, 250) impresión (tesla_model_s.number_of_wheels) - 4tesla_model_s.number_of_wheels - 2 - establecer el número de ruedas a #2 Esto es ligeramente diferente de la definición del método. El método aquí se basa en la propiedad. Por ejemplo, cuando establecemos el nuevo número de neumáticos, no consideramos estos dos como parámetros, pero establecemos el valor 2 en number_of_wheels. Esta es una forma de escribir un captador de estilo python y un establecedor de código. Pero también podemos usar este método para otras cosas, como el método make_noise. Veamos: categoría del vehículo: __init__ def (__init__, number_of_wheels, type_of_tank, seating_capacity, maximum_velocity): self.number_of_wheels number_of_wheels self.type_of_tank type_of_tank self.seating_capacity seating_capacity self.maximum_velocity maximum_velocity def make_noise(self): print('VRUUUUUUUM') Cuando llamamos a este método, simplemente devuelve una cadena VRRRRURUUUUM. tesla_model_s - Vehículo(4, 'eléctrico', 5, 250) tesla_model_s.make_noise() - Paquete Python VRUUUUUM: Ocultar información La encapsulación es un mecanismo que restringe el acceso directo a datos y métodos de objetos. Al mismo tiempo, sin embargo, facilita el manejo de datos (métodos de objeto). Package se puede usar para ocultar los miembros de datos y las funciones miembro de acuerdo con esta definición, esto significa que el paquete. Objetos dentro de una vista externa que normalmente están ocultos en la definición del objeto. – Wikipedia Todas las representaciones internas de objetos se ocultan desde el exterior. Solo el propio objeto puede interactuar con sus datos internos. En primer lugar, necesitamos entender cómo funcionan las variables y los métodos de presencia abierta y cerrada. Variables de presencia pública Para la clase Python, podemos preparar una variable de presencia pública en el método de fabricación. Echemos un vistazo a esto: En este método de fabricación: categoría Persona: def __init__ (auto, first_name): self.first_name first_name Aquí, aplicamos el valor first_name como parámetro a las variables de presencia pública. tk - Persona ('TK') print(tk.first_name) ? ?> TK En clase: clase Persona: first_name 'TK' Aquí, no es necesario first_name como parámetro, todos los objetos de instancia tienen un atributo de clase que se inicializa con TK. tk - Person() print(tk.first_name) ? > TK Muy cool, y ahora hemos aprendido que podemos usar variables públicas de instancias y. Otra cosa interesante de la parte pública es que podemos gestionar los precios de las variables. ¿De qué estoy hablando? Nuestro objeto puede gestionar sus valores variables: Descargar y establecer valores de variables. Todavía en la Persona de la Categoría, queremos establecer otro valor para la variable first_name: tk - Persona ('TK') tk.first_name 'Kaio'print(tk.first_name) ?> Kaio Esto está bien, acabamos de establecer otro valor (kaio) para la variable de presencia first_name y actualizar Precio. Es así de simple. Debido a que esta es una variable pública, podemos hacer eso. Python No público variables No usamos el término privado aquí porque todas las propiedades en Python no son realmente privadas (normalmente no hay una cantidad innecesaria de trabajo). - PEP 8 Como variable de presencia pública (variables de presencia pública), podemos definir una variable de presencia no pública (variables de presencia no pública) dentro del fabricante o clase. La diferencia en la sintaxis es que para las variables de instancia no públicas (variables de instancia no públicas), se utiliza un carácter de subrayado (_) antes del nombre de la variable. Las variables de casos privados 'que no son accesibles desde dentro del objeto no existen en Python, sin embargo, hay un contrato que la mayoría del código Python seguirá: los nombres de subrayado (como _spam) se considerarán como una parte no pública de la API (ya sea función, método o miembro de datos) Aquí está el código de ejemplo: categoría Persona: def __init__ (solo, first_name, correo electrónico): self.first_name first_name self._email correo electrónico ¿Ha visto la variable de correo electrónico? Así que definimos variables no públicas: podemos acceder a ellas y actualizarlas. Las variables no públicas son simplemente una cuestión de uso y deben tratarse como una parte no pública de la API. Por lo tanto, usamos un método dentro de la definición de clase para implementar esta funcionalidad. Vamos a aplicar dos métodos (e-mail y update_email) para profundizar en nuestra comprensión: categoría Persona: def __init__ (auto, first_name, correo electrónico): self.first_name - first_name self._email - e-mail def update_email (self, new_email): self._email - new_email def email (auto): return self._email Ahora podemos usar estos dos métodos para actualizar y acceder a variables no públicas. Este es un ejemplo Hemos preparado un nuevo objeto usando first_name TK y correo electrónico Use el método para acceder a un correo electrónico de variable no público y salga ¡Intente configurar un nuevo mensaje de correo electrónico fuera de la categoría Debemos tratar las variables no públicas como partes no públicas de la API Use nuestro método de presencia para actualizar las variables Success no públicas! Lo actualizamos en el aula utilizando métodos auxiliares. Método público de Python para métodos públicos, también podemos usarlos en categorías: categoría Persona: def __init__ (__init__ de nosotros, first_name, edad): self.first_name de first_name self._age de la edad show_age (auto): devolución self._age Vamos a probarlo: tk - Cara ('TK', 25) imprimir (tk.show_age()) - > 25 Eso está bien - no tenemos ningún problema de uso en nuestra clase. Métodos no públicos de Python Pero con un método no público, no podemos hacerlo. Si queremos aplicar la misma clase ahora usamos el método de subrayado (_) show_age público. Categoría Persona: def __init__ (first_name, edad): self.first_name - first_name self._age - edad def _show_age (auto): regreso self._age Ahora, intentaremos probar llamar a este método no público con nuestro objeto: tk - Persona ('TK', 25) print(tk._show_age()) ? > 25 Podemos acceder a él y actualizarlo. El método no público es simplemente un contrato y debe considerarse una parte no pública de la API. Este es un ejemplo de cómo podemos usarlo: categoría Persona: def __init__ (self, first_name, edad): self.first_name de la first_name self._age de la edad show_age (auto): devolución self._get_age() def _get_age(self): return self._age tk á Person ('TK', 25) print(tk.show_age()) ? > 25 Aquí hay un método _get_age no público y un método público show_age. show_age puede ser utilizado por nuestro sujeto (no en nuestra clase), mientras que _get_age se utiliza sólo en nuestra definición de clase (en el show_age). Pero de nuevo, esa suele ser la práctica. Resumen del paquete de Python A través del paquete, podemos asegurarnos de que la representación interna del objeto esté oculta desde el exterior. Transferencia de Python: Comportamiento y características Algunos objetos tienen algo en común: su comportamiento y características. Por ejemplo, heredé algunos de los rasgos y comportamientos de mi padre. Heredé las características de sus ojos y cabello, así como su impaciencia y comportamiento introvertido. En la programación orientada a objetos, una clase puede heredar los atributos comunes (datos) y comportamientos (métodos) de otra clase. Echemos un vistazo a otro ejemplo y aplíquelo a Python. Imagina el auto. El número de ruedas, la capacidad del asiento y la velocidad máxima son todas las características de un coche. Se puede decir que la clase ElectricCar hereda estas mismas propiedades de la categoría normal __init__. number_of_wheels, seating_capacity, maximum_velocity): self.number_of_wheels number_of_wheels self.seating_capacity seating_capacity self.maximum_velocity maximum_velocity La implementación de nuestra categoría de automóviles: impresión my_car (my_car, 5, 250) (my_car.number_of_wheels) (my_car.seating_capacity) (my_car.maximum_velocity) Una vez preparada, podemos utilizar todas las variables de ejemplo creadas. ¡Terrible!. En Python, heredamos la clase primaria como argumento secundario. Una clase De ElectricCar puede heredar nuestra categoría de automóviles. Categoría ElectricCar (Coche): def __init__ (auto, number_of_wheels, seating_capacity, maximum_velocity): Car.__init__ (auto, number_of_wheels, seating_capacity, maximum_velocity) Es tan simple. No es necesario aplicar ningún otro método porque esta clase ha completado la herencia de la clase primaria (heredada de la clase car). Vamos a probar: my_electric_car de impresión ElectricCar(4, 5, 250) •> 4 impresión (my_electric_car.seating_capacity) á > 5 impresión (my_electric_car.maximum_velocity) á > 250 Eso es hermoso. Fuente original. Otros tutoriales de Python ¿Qué es Python - A A Ventajas de la Guía de Programación de Python

pabucumun ajanı 2 pdf
cuidados de enfermeria mediatos del recien nacido
besame mucho easy piano pdf
step-brother 2016 movie
best places to download cc images
calculus 1 made easy pdf
dark souls artorias dlc guide
holland code personality test pdf
desarrollo y bienestar social pdf
can_and_could_exercises_for_elementary.pdf
reporting_verbs_exercises_fce.pdf
55116422020.pdf