Programador Junior en Machine Learning I

Argentina programa 4.0

Ministerio de Desarrollo Productivo Argentina

Secretaría de Economía del Conocimiento

Semana 2

- **Conceptos básicos de Python**
  - Sintaxis básica
  - Creación de variables
  - Estructuras de control
  - Tipos de datos
  - Operadores
  - Uso de un IDE (Entorno de Desarrollo Integrado)
- **Instalación de Visual Studio Code**

Conceptos básicos de Python

Python es un lenguaje de programación interpretado (esto es: se ejecuta en el momento línea por línea) que está cobrando una popularidad inmensa gracias a sus aplicaciones en el ámbito de la IA.

```
# Esto es un comentario

def saludame():  # una función que se llama "saludame"
    print("Hola gente!")  #imprime en consola



saludame()        # llama a la función "saludame"
```

Escribir código en Python es lo más sencillo del mundo. No hay que preocuparse por cierres de líneas o cierres de bloques, solamente respetar el espaciado/sangrías para definir su estructura.

Variables en Python

Una variable se puede considerarla una “caja” donde podemos guardar valores u objetos de todo tipo. Pueden tener el nombre que uno desee, con la salvedad de que no comiencen con un número, que no tengan espacios y que el único símbolo admitido es el guión bajo (_).

Podemos hacer operaciones matemáticas con estas variables y guardar los resultados de las operaciones en otras variables, o incluso reemplazar el valor de la variable actual.

```
# Ejemplo de variable

x = 5    # una variable que se llama “x” y que guarda un 5
y = 4    # una variable “y” que guarda un 4
z = x + y    # una variable “z” donde guardo la suma

print(z)     # imprime en consola el valor actual de “z”

saludo = “Bienvenidos al curso!”  # variable cadena de
texto
print(saludo)
```

Funciones en Python

Una función es un bloque de código que tiene un nombre, y se ejecuta donde quiera que se llame a su nombre. Se lo define con la palabra clave “def”. Al contenido de la función se lo antepone de una sangría.

Una función puede devolver un valor que luego lo podemos guardar en una variable, o puede devolver nada. Para que nos devuelva un valor, debemos usar la palabra clave “return”. Las funciones que no utilizan return no pueden guardar sus valores en una variable.

```
# Función que no devuelve un valor

def saludame(): # una función que se llama “saludame”
    print(“Hola gente!”) #imprime en consola

saludame()        # llama a la función “saludame”

# Función que devuelve un texto
def saludalo(): # una función que se llama “saludalo”
    return “Hola gente!” #solo devuelve el texto

mensaje = saludalo() # llama a la función
print(mensaje)    # imprime lo que retornó “saludalo”
```

Operadores aritméticos en Python

Los operadores en Python son símbolos que permiten realizar diferentes tipos de operaciones sobre los datos, como aritméticas, de asignación y de comparación. Estos son algunos de los operadores más comunes en Python:

Operadores aritméticos: son los que realizan operaciones matemáticas sobre los números, como la suma (+), la resta (-), la multiplicación (*), la división (/), el módulo (%), la potencia (**), y la división entera (//).

```
# Ejemplos operadores aritméticos

a = 10
b = 3
c = a + b # Suma
d = a - b # Resta
e = a * b # Multiplicación
f = a / b # División
g = a % b # Módulo
h = a ** b # Potencia
i = a // b # División entera
```

Operadores de asignación en Python

Operadores de asignación: son los que asignan un valor a una variable, como el igual (=), el más igual (+=), el menos igual (-=), el por igual (*=), el entre igual (/=), el módulo igual (%=), el potencia igual (**=), y la división entera igual (//=). Por ejemplo:

```
# Asigna el valor 10 a la variable "a"
a = 10
# Suma 5 al valor de a y lo asigna a la misma variable
a += 5
# Resta 2 al valor de a y lo asigna a la misma variable
a -= 2
# Multiplica por 3 el valor de a y lo asigna a la misma variable
a *= 3
# Divide entre 2 el valor de a y lo asigna a la misma variable
a /= 2
# Calcula el módulo de 4 del valor de a y lo asigna a "a"
a %= 4
# Eleva al cuadrado el valor de a y lo asigna a la misma variable
a **= 2
# Calcula la división entera de 3 del valor de a y lo asigna a "a"
a //= 3
```

Operadores de comparación en Python

Operadores de comparación: son los que comparan dos valores y devuelven un valor booleano (True o False) según el resultado de la comparación, como el mayor que (>), el menor que (<), el igual que (==), el mayor o igual que (>=), el menor o igual que (<=), y el distinto que (!=). Por ejemplo:

```
a = 10
b = 3
c = a > b # True, porque 10 es mayor que 3
d = a < b # False, porque 10 es menor que 3
e = a == b # False, porque 10 no es igual que 3
f = a >= b # True, porque 10 es mayor o igual que 3
g = a <= b # False, porque 10 es menor o igual que 3
h = a != b # True, porque 10 es distinto que 3
```

Operadores lógicos en Python

Los operadores lógicos en Python son palabras clave que se utilizan para combinar o negar expresiones booleanas. Los operadores lógicos en Python son and, or y not. Estos operadores tienen la siguiente sintaxis y significado:

- and: devuelve True si ambos operandos son True, y False en cualquier otro caso.

- or: devuelve True si al menos uno de los operandos es True, y False solo si ambos operandos son False.

- not: devuelve el valor contrario del operando.

Los operadores lógicos se pueden utilizar para construir condiciones complejas a partir de expresiones simples.

```
if x > 0 and x < 10:
    print("x está entre 0 y 10")
# Los operadores lógicos también se pueden combinar entre sí
# usando paréntesis para indicar el orden de evaluación.
# Acá comprobamos si "y" es par y múltiplo de 5
if (y % 2 == 0) or (y % 5 == 0):
    print("y es par o múltiplo de 5")

# En general cualquier valor que sea cero, vacío o None se
# considera falso. Por ejemplo:
if "" or 0 or None:
    print("Esta línea no se ejecuta")
if "hola" and [1, 2, 3] and 42:
    print("Esta línea sí se ejecuta")
# Los operadores lógicos devuelven el valor del último
# operando evaluado, no necesariamente un valor booleano.
a = 10
b = 20
c = a and b # c vale 20
d = a or b # d vale 10
e = not a # e vale False
```

Condicionales en Python

Una estructura de control condicional es una estructura de código que ejecuta una parte del código u otra dependiendo si se cumple o no una condición.

Se utiliza la palabra clave “if” para evaluar la condición que puede resultar en verdadera o falsa. Si es verdadera, ejecuta la primera parte. Si es falsa, ejecuta la parte definida por la palabra clave “else”.

Dicho esto, como condición también podemos poner variables que sean True o False, o incluso funciones que retornan True o False.

```
# Ejemplo con una condición

x = 6

if x > 5:
    print(“El número es mayor que 5”)  # Viene acá
else:
    print(“El número no es mayor que 5”)

# Ejemplo con una variable booleana

te_dejo_pasar = False

if te_dejo_pasar:
    print(“Pasaste”)
else:
    print(“No pasaste”)   # Viene acá
```

Bucles en Python

Los ciclos son estructuras de control que permiten repetir un bloque de código mientras se cumpla una condición o mientras se recorre una secuencia de elementos. En Python hay dos tipos de ciclos: while y for. El ciclo while se usa para repetir un bloque de código mientras una condición sea verdadera, mientras que el for se ejecuta un número fijo de veces.

El while se ejecuta por medio de una condición, de manera que siempre hay que buscar la forma de que esa condición se actualice una vez alcanzado el objetivo, sino el bucle será infinito y eso es algo que no deseamos.

```
# El while se va a repetir hasta que "contador"
# sea mayor que 10

contador = 1

while contador <= 10:  # mientras sea menor o igual a 10
    print(contador)
    contador = contador + 1

# El for se va a repetir 10 veces fija.
# El contador se define dentro del mismo for
# sin necesidad de una variable

for numero in range(1, 11):
    if numero % 2 == 0:   # comprueba si "numero" es
par
        print(numero)
```

Tipos de datos en Python

Los tipos de datos básicos en Python son aquellos que definen un conjunto de valores con ciertas características y propiedades. Entre ellos se encuentran los enteros, los flotantes, los booleanos y las cadenas de texto.

Los enteros son números sin parte decimal, como 1, -5 o 42.

Los flotantes son números con parte decimal, como 3.14, -2.5 o 6.0.

Los booleanos son valores lógicos que pueden ser True (verdadero) o False (falso).

Las cadenas de texto son secuencias de caracteres, como "Hola", "Python" o "Tipos de datos".

```
# Ejemplo de operaciones con enteros
a = 10
b = -3
c = a + b
print(c) # imprime 7

x = 2.5
y = -1.2
z = x * y
print(z) # imprime -3.0

a = True
b = False
c = a and b
print(c) # imprime False

s = "Tipos de datos"
t = s.upper()    # se le aplica la función “upper” a “s”
print(t) # imprime TIPOS DE DATOS
```

Estructuras de datos en Python

Python también ofrece estructuras de datos que pueden contener múltiples valores en una sola variable, como las listas, las tuplas y los diccionarios.

Las listas son secuencias ordenadas y mutables de elementos, que pueden ser de cualquier tipo. Se definen con corchetes y se separan los elementos con comas.

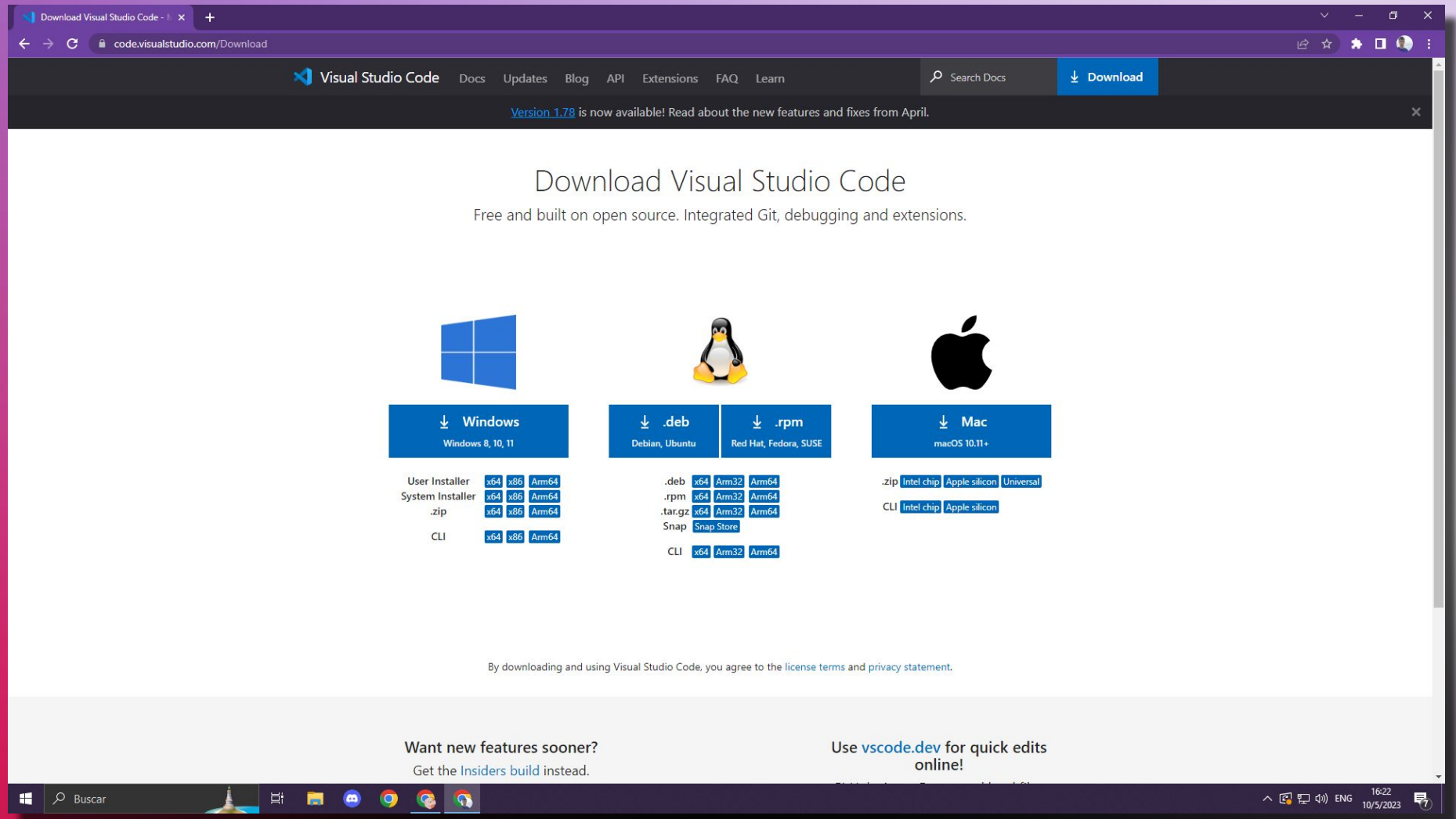
Las tuplas son secuencias ordenadas e inmutables de elementos, que también pueden ser de cualquier tipo.

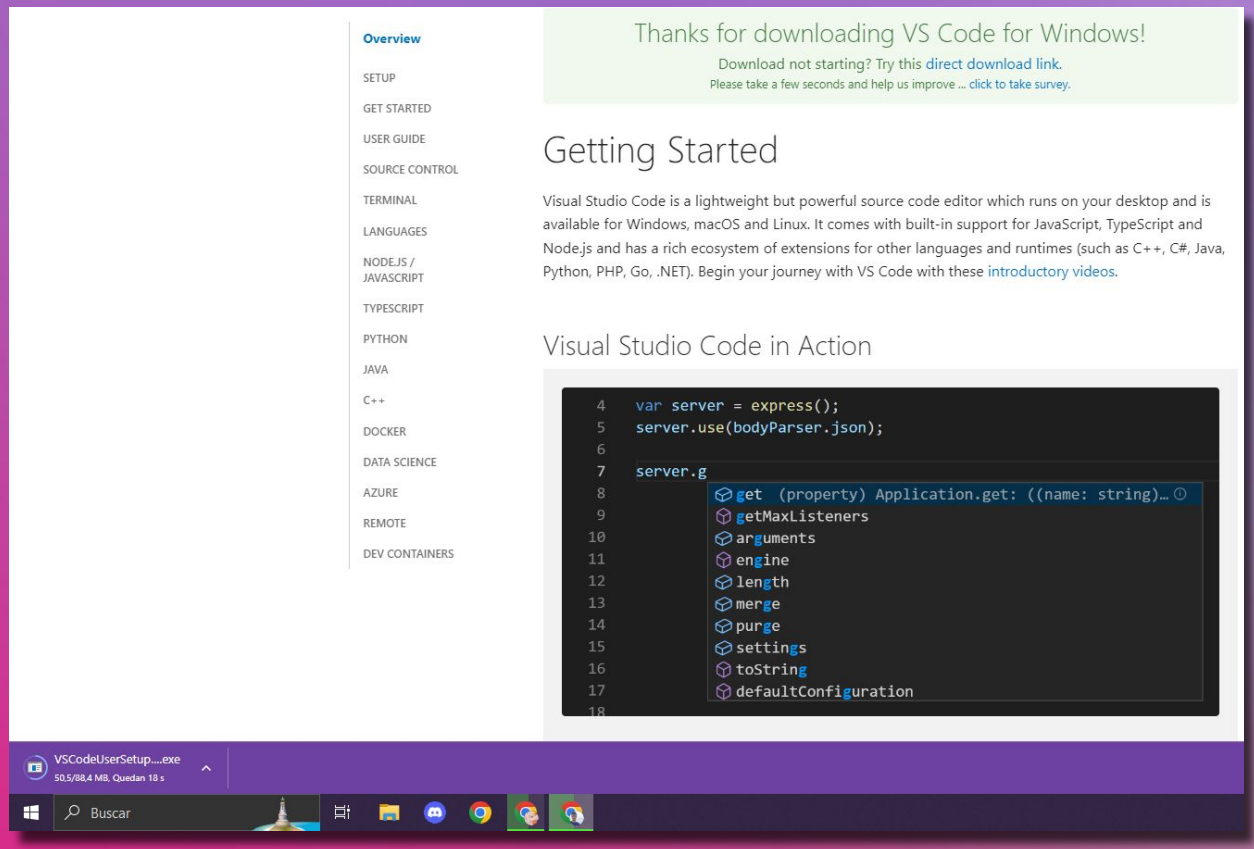
Los diccionarios son colecciones no ordenadas y mutables de pares clave-valor, donde las claves deben ser objetos inmutables y únicos, y los valores pueden ser de cualquier tipo.

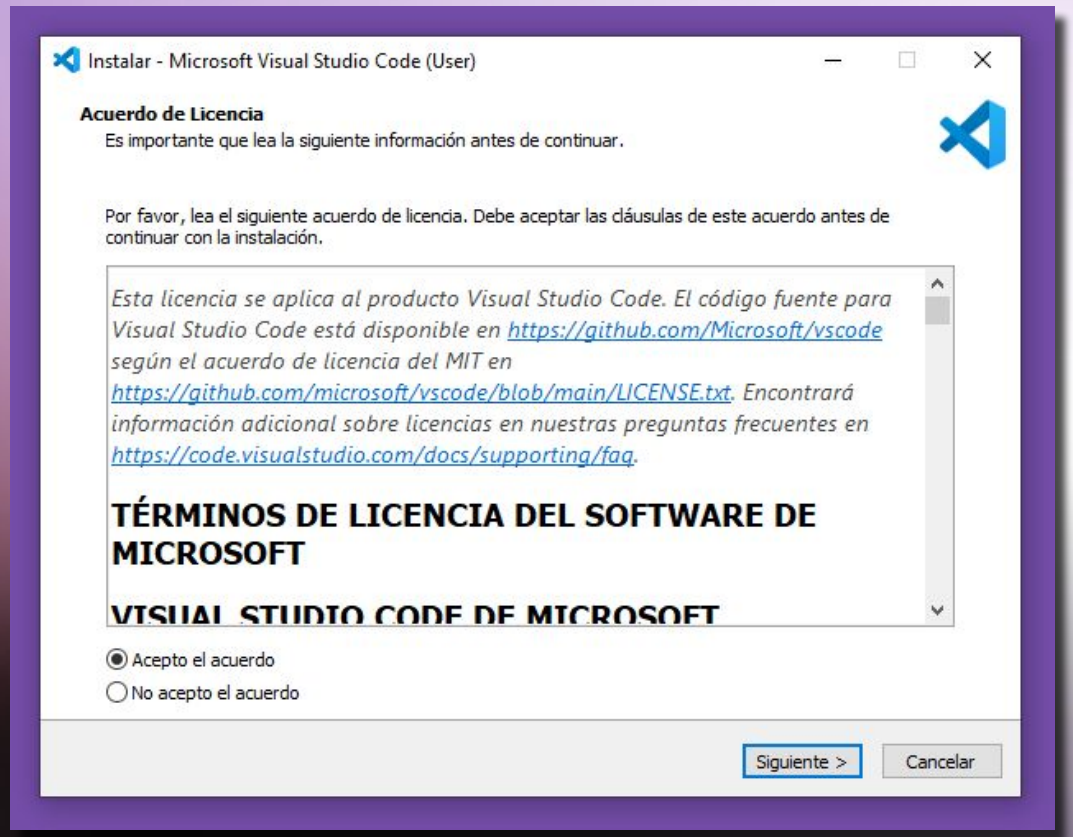
```
# Listas
numeros = [1, 2, 3, 4, 5]
nombres = ["Ana", "Juan", "Sofía", "Pablo"]
mezcla = [True, 10.5, "abc", [0, 1, 1]]
numeros.append(6) # Añade el 6 al final de la lista
nombres.remove("Ana") # Elimina el elemento "Ana"
print(mezcla[2]) # Imprime el tercer elemento de la lista: "abc"

#Tuplas
colores = ("Azul", "Verde", "Rojo", "Amarillo")
coordenadas = (10, 20)
vacio = ()
print(colores[0]) # Imprime el primer elemento de la tupla: "Azul"
print(coordenadas[-1]) # Imprime el último elemento de la tupla: 20

# Diccionarios
edades = {"Ana": 25, "David": 18, "Lucas": 35}
productos = {1: "Lápiz", 2: "Goma", 3: "Regla"}
vacio = {}

edades["Ximena"] = 30 # Añade un nuevo par al diccionario
productos.pop(2) # Elimina el par con la clave 2 del diccionario
print(vacio.get("clave")) # Imprime el valor asociado a la clave
"clave" o None si no existe
```

Instalación de Visual Studio Code

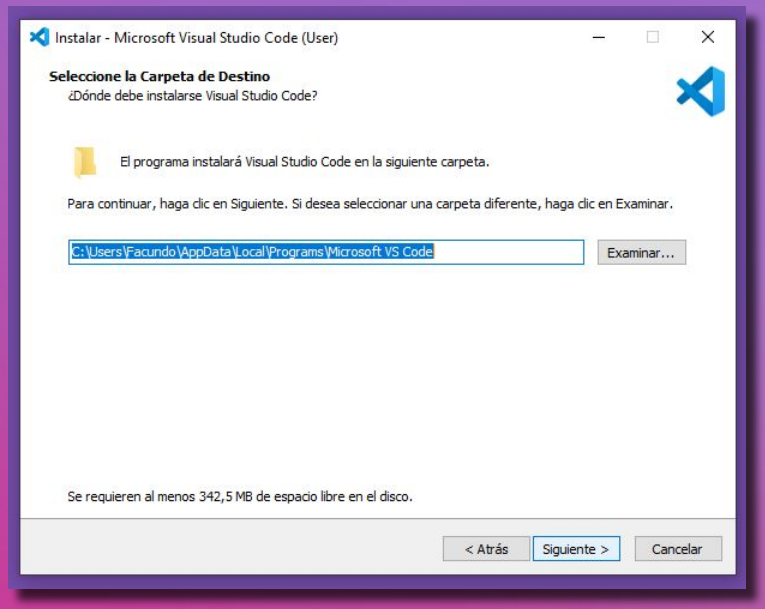
1. Ingrese al siguiente enlace: https://code.visualstudio.com/Download
2. Elija la opción según su sistema operativo.

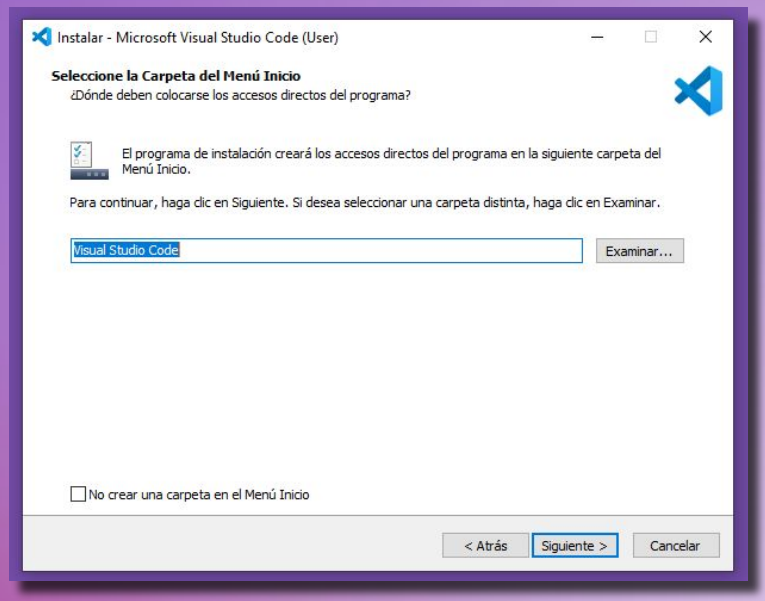


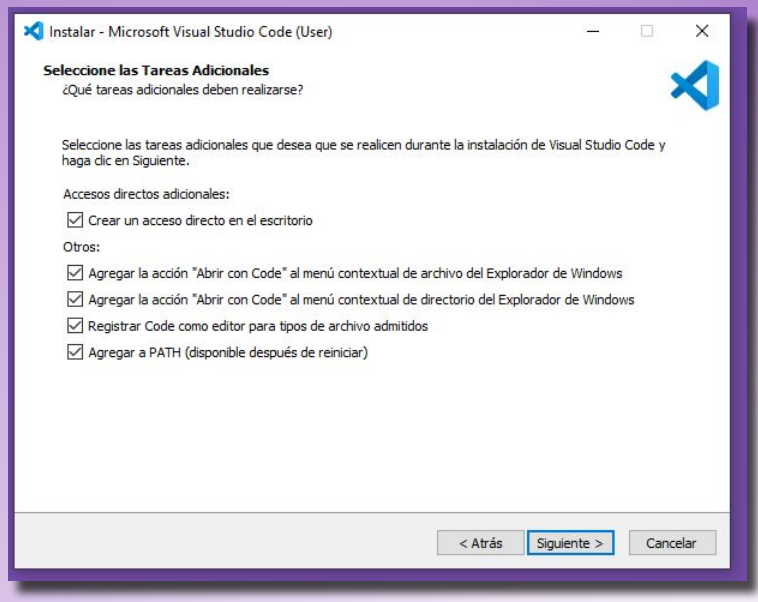




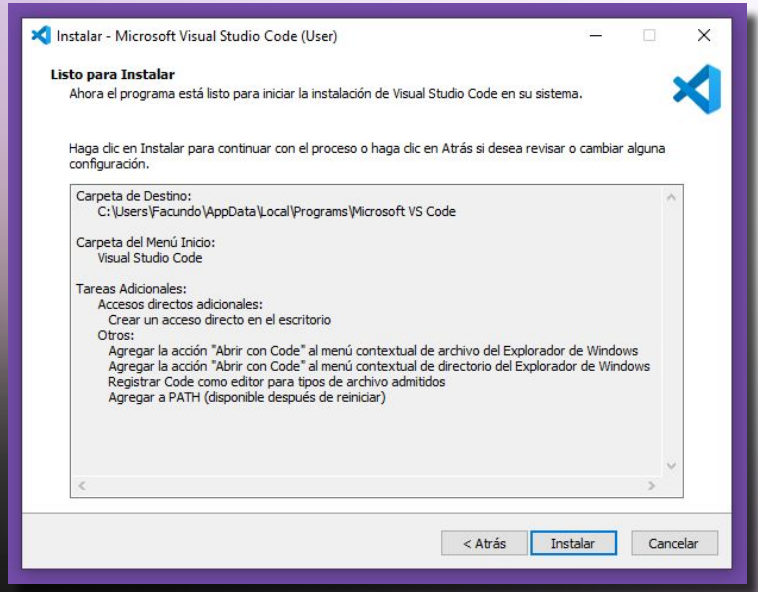
Espere que termine la descarga y ejecute el programa.

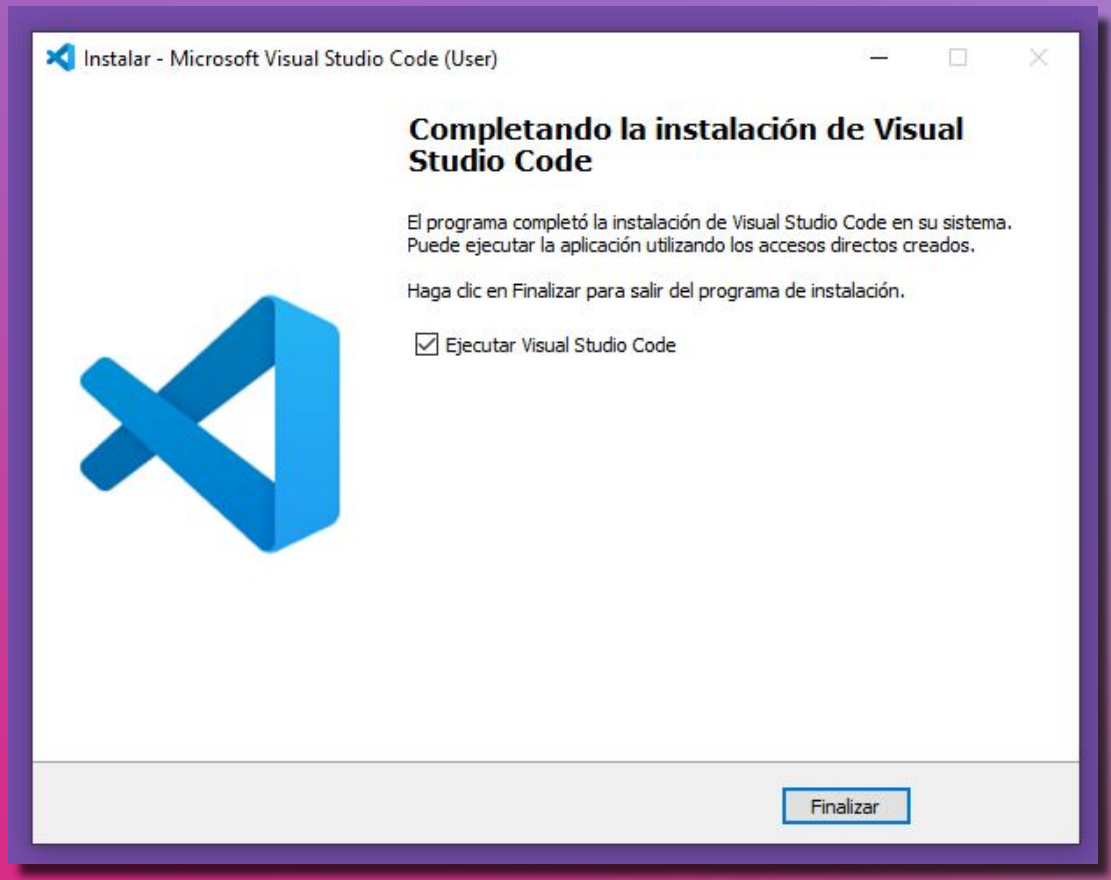


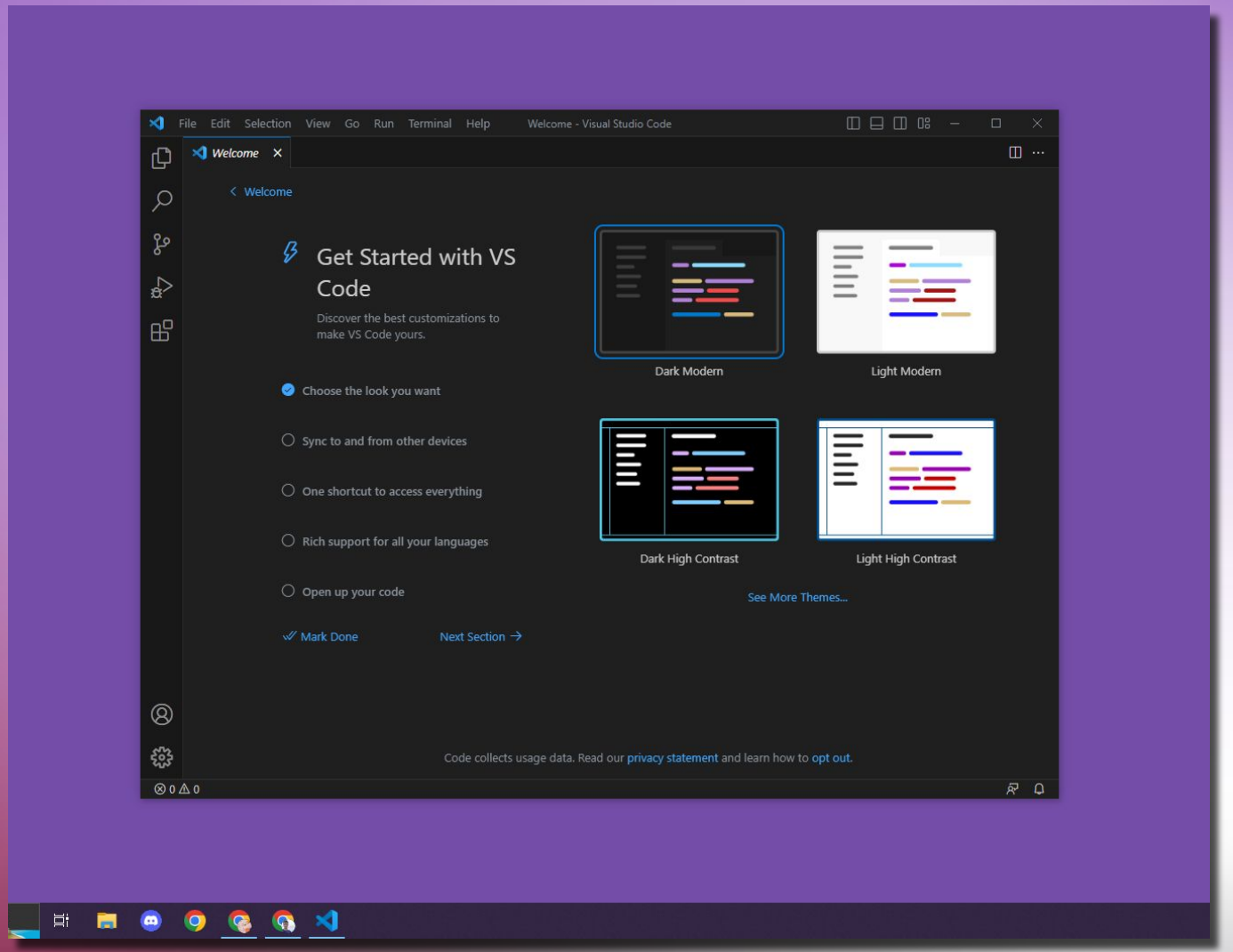




Simplemente darle a “Siguiente” en cada ventana e “Instalar”.







Descarga e instalación finalizada.