Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-2

Publicación: 14 de agosto de 2025
Actividad evaluada

Actividad 1
Entorno de trabajo

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC1`
- **Fecha máxima de entrega:** Jueves 14 de agosto 17:20
- **Ejecución de actividad**: La Actividad será ejecutada **únicamente** desde la terminal del computador. **Los *paths* relativos utilizados en la Actividad deben ser coherentes con esta instrucción, y no pueden modificarse.**

Objetivos de la actividad

- Familiarizarse con los comandos de Git.
- Crear, modificar y actualizar el repositorio personal.
- Crear y modificar archivos en Python (.py).

Flujo del programa

Esta actividad consta de 3 partes principales, las cuales se dividen en diversos temas:

Parte 1. Antes de empezar

Parte 2. Actividad

Parte 3. Sigamos practicando

Cada una de estas partes se introducirán diversas herramientas del curso.

La única parte que es evaluada es la **Parte 2**, pero pese a eso recomendamos seguir cada parte a conciencia ya que les ayudarán a aprender a utilizar Git, la terminal, y cómo ejecutar tests.

Antes de empezar

1. Conociendo a IIC2233

Esta es la primera actividad formativa en IIC2233. El objetivo de esta primera parte es entender cuáles son las principales páginas relacionadas con el curso. Los requisitos para empezar son:

1. Haber instalado Git en tu computador (más información en este enlace).
2. Haberte registrado en GitHub y completado el formulario de Creación de Repositorio.
3. Una vez completado el formulario, te debió llegar un correo de GitHub con una invitación a un repositorio en la organización del curso (revisa bien tu correo utilizado para la cuenta de GitHub).

1.1. *Syllabus* (`syllabus`)

El *Syllabus* es un **repositorio** de GitHub donde, como equipo docente, subiremos todos los enunciados de las tareas, actividades y ayudantías. Puedes ver el *Syllabus* en `www.github.com/IIC2233/syllabus`.

1.2. Repositorio de apuntes (`contenidos`)

Los apuntes con los contenidos del curso se encontrarán en un repositorio llamado `contenidos`, al que puedes llegar haciendo clic en un vínculo ubicado en la página principal del curso, o yendo directamente a `www.github.com/IIC2233/contenidos`.

1.3. Tu repositorio personal

Luego de registrarte en el curso, se te ha creado un repositorio **personal y privado** donde deberás trabajar y entregar tus actividades y tareas. Este se ubica en:

`www.github.com/IIC2233/usuario-iic2233-2025-2`

donde debes reemplazar "`usuario`" por tu usuario de GitHub. Por ejemplo, si tu usuario es "`pepa`", tu repositorio estará en:

`www.github.com/IIC2233/pepa-iic2233-2025-2`

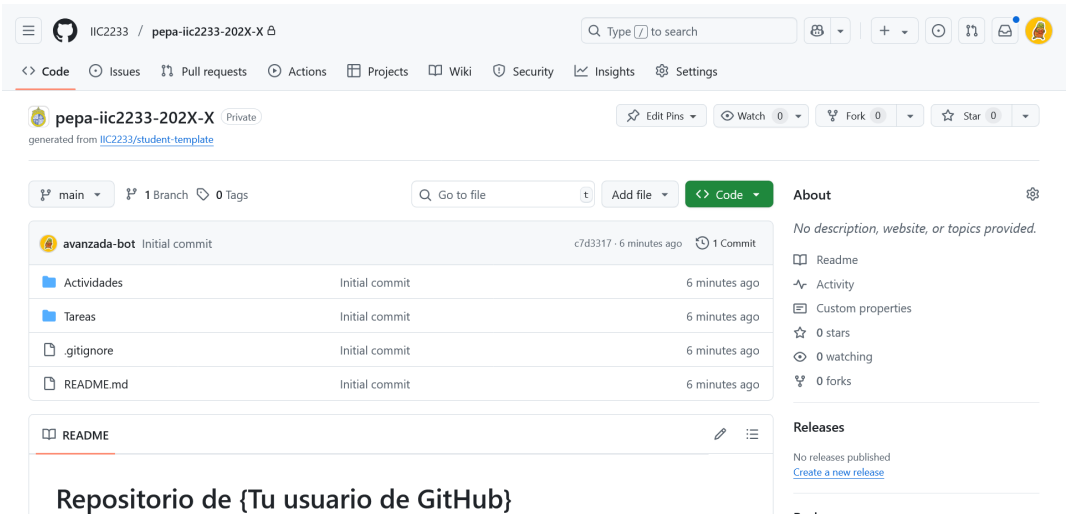


Figura 1: Ejemplo de repositorio personal

2. Llevando IIC2233 a tu PC gracias a `git`

El objetivo de esta parte es tener un primer contacto con `git`, el sistema de control de versiones que se utiliza en este curso.

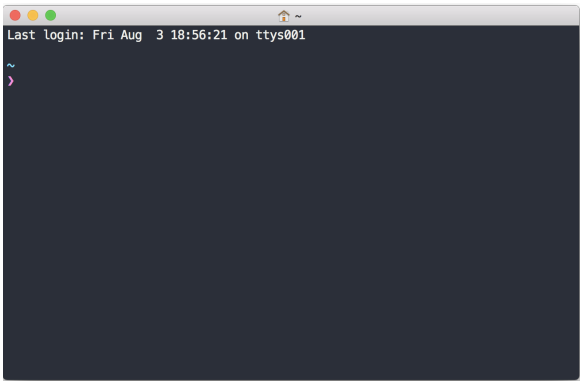
2.1. Abrir la terminal

Necesitamos aprender a navegar por las carpetas del computador usando la consola. En esta parte de la guía, sigue todos los pasos y verifica que te sale lo mismo que está aquí.

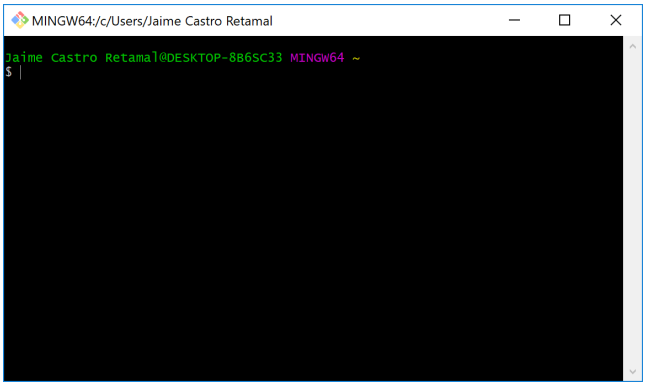
Para abrir la consola:

- **macOS o Linux:** busca el programa "`Terminal`" o similar.
- **Windows:** busca el programa "`Git Bash`". Este programa es una consola que además implementa algunos de los comandos que podrías encontrar en Linux, por lo que es mucho más amigable que el "Símbolo del Sistema".

La consola en todo momento está ubicada en una carpeta (o directorio). Al abrir la consola, ésta se abre en la carpeta de tu computador que contiene tus datos. Esta carpeta suele llamarse "`home`" o "∼". Dentro de ella, suele encontrarse el escritorio o la carpeta de documentos.



(a) Consola abierta en macOS



(b) Consola abierta en Windows

Luego, puedes usar el comando `cd` para moverte por diferentes carpetas. Recomendamos revisar los contenidos de terminal de la semana 1 para aprovechar lo que más puedas la terminal. ¡Esta la usarás durante todo el semestre!

2.2. Clonar repositorio personal

En esta parte, clonaremos tu repositorio personal para que puedas entregar tus actividades y tareas. Para esto, primero necesitamos generar un *token* especial, que nos servirá como contraseña para poder acceder a cualquier repositorio que no sea público, por ejemplo, tu repositorio personal.

Para esto, primero debes acceder a este enlace e ingresar con tu cuenta de Github. Verás una vista como la Figura 3.

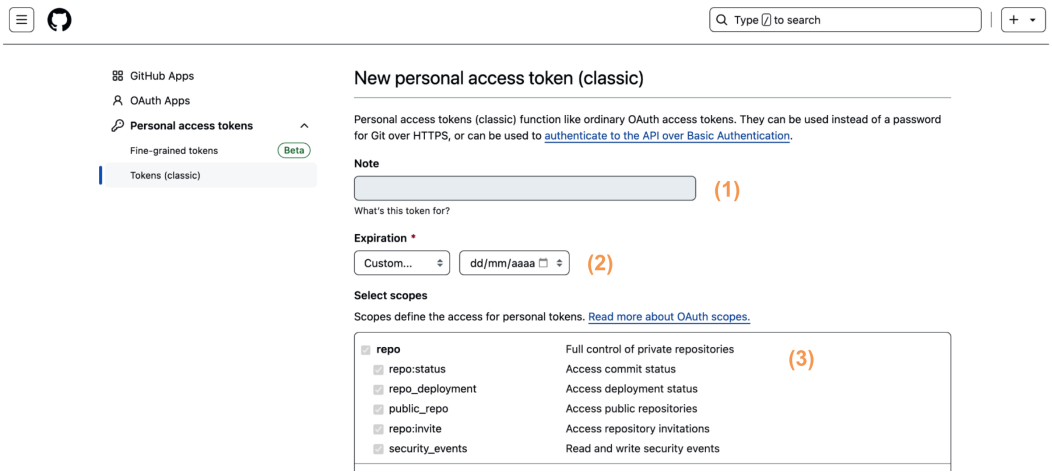


Figura 3: Interfaz para generar *token* personal

Luego, debes completar este formulario del siguiente modo:

1. Agregar una descripción en la sección de *"Note"*. Este será un texto para darle una descripción al *token* generado.
2. Definir una fecha de expiración. Se recomienda poner una posterior al cierre de semestre. Una vez pase la fecha definida, el *token* ya no servirá y deberás generar uno nuevo.
3. En la sección de *"Select scopes"*, haz *click* en cada cuadrado. Así haremos un *token* con los permisos asociados a *"repo"* (*repo:status, repo_deployment, public_repo, repo:invite, security_events*). Más adelante, en la carrera, aprenderás a generar un *token* con permisos más exclusivos.

Tras cumplir con los 3 pasos, debes hacer *click* en un botón que está al final del formulario que dice "***Generate token***". La vista de tu navegador se actualizará y te aparecerá un texto que comienza con `"ghp_"`. Ese será tu *token* que deberás guardar y usar siempre en tu terminal para interactuar con tu repositorio privado (*clone, pull, push*). No pierdas este *token* porque solo aparecerá una vez al momento de generarlo. En caso de olvidarlo, deberás crear uno nuevo siguiendo todos los pasos anteriores.

Ahora que ya tienes tu *token* personal generado. Podemos empezar con el proceso de clonar tu repositorio.

1. Asegúrate de que la terminal esté dentro de la carpeta donde quieras tener tu repositorio personal, como el escritorio o alguna carpeta propia. Si no, navega usando los comandos mencionados hasta encontrarlo.
2. Ve a la página de tu repositorio y copia el vínculo que permite clonar el repositorio (busca el botón verde que resalta y que muestra la Figura 4).
3. En la consola, ejecuta el comando `git clone url_copiada` para clonarlo.
4. Se te pedirá ingresar tu usuario y luego el *token* que acabamos de generar. Este **no se verá en tu terminal al momento de pegarlo**, pero es normal porque es por temas de seguridad. Luego presionas *enter* y se comenzará a clonarse tu repositorio.
5. Deberías ver que se creó la carpeta con el contenido de tu repositorio personal[1].

[1]Si no sabes dónde se creó, recuerda revisar los contenidos de terminal, en ellos se indica un comando que muestra el directorio de trabajo actual.

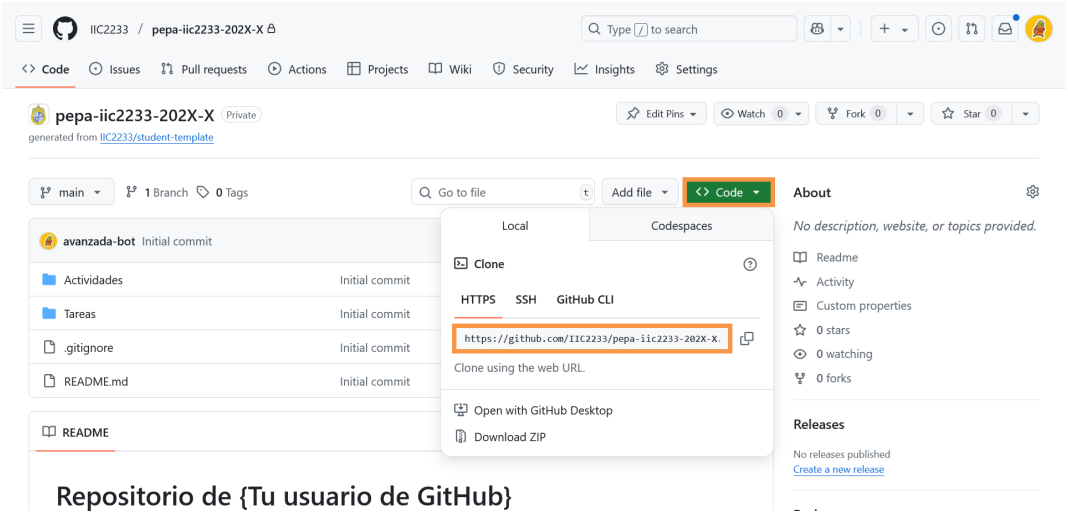


Figura 4: Donde encontrar enlace de repositorio.

En caso de que ya hayas copiado tu repositorio personal en un lugar que no te guste, simplemente puedes mover la carpeta completa a un directorio que prefieras, los archivos que permiten que `git` funcione se encuentran dentro de tu carpeta y seguirán funcionando normalmente.

2.3. Clonar otros repositorio importantes del curso

Aprovechando el vuelo, repite los mismos pasos del punto anterior pero para `Syllabus` y para `contenidos`, ya que tendrás que estar pendiente de estos repositorios durante el semestre.

Actividad

¡Ahora sí! Para empezar a trabajar, haremos una pequeña actividad para familiarizarnos con las herramientas del curso.

1. El primer *add*, *commit* y *push*

Haremos nuestro primer *commit* dentro del repositorio personal. Para trabajar en él, tu consola deberá estar **dentro** de la carpeta del repositorio (podrías usar el comando `cd`).

1. Dentro del repositorio, hay un archivo llamado `README.md` creado. En este, hay información sobre el uso de tu repositorio, y dejamos espacios para que rellenes con tus datos. Abre el archivo en tu computador con algún editor de texto y rellena los espacios con tus datos.

   Si rellenaste la información antes de la actividad, realiza un cambio que afecte el archivo. Puedes agregar emojis, poner tu sobrenombre o lo que se te ocurra que implique un cambio en el archivo.

2. Ahora vuelve a la consola. Revisa el estado del repositorio utilizando `git status`. Observa que tu archivo recién editado aparece listado como *"modified"* en *"Changes not staged for commit"*.

3. Agrega el archivo `README.md` al *stagging area* mediante el comando: `git add README.md`.

4. Revisa el estado del repositorio (`git status`) y verifica que tu archivo recién editado aparece listado como *"modified"* en *"Changes to be committed"*.

5. Utiliza: `git commit -m "README actualizado"`. El texto entre comillas se conoce como mensaje del *commit* y debe describir lo que fue agregado mediante `git add`.

6. Vuelve a revisar el estado (`git status`) del repositorio nuevamente. No deberías ver listado a `README.md`, pero si un mensaje que dice *"Your branch is ahead of 'origin/main' by 1 commit."*, lo cual significa que el *commit* fue creado correctamente,

7. Ahora, para escribir tus cambios en GitHub utiliza: `git push`.

8. Vuelve a revisar el estado del repositorio (`git status`), debe decir: *Your branch is up to date with 'origin/main'. nothing to commit, working tree clean.*

9. Después ve el contenido del repositorio en un navegador web (GitHub) y verifica tus cambios. Si los ves, ¡lo lograste!

Importante

Si hiciste todo lo anterior bien, deberían ver 1 nuevo *commit* en la página de Github y los cambios aplicados en el tu `Readme.md`.

2. Crear y ejecutar archivo `.py` desde la terminal

Crearemos nuestro primer archivo `.py`. Primero deberás ubicar tu terminal dentro de la carpeta "`Actividades/AC1`" y crear un archivo llamado "`mi_primer_archivo.py`". Recuerda que existen comandos de terminal para lograr esto:

- Windows: `echo > archivo.txt` creará un archivo vacío cuyo nombre es "`archivo`" y su extensión es "`txt`".
- MacOS o Linux: `touch archivo.txt` creará un archivo vacío cuyo nombre es "`archivo`" y su extensión es "`txt`".

Luego, utilizando tu herramienta de desarrollo, deberás abrir y editar el archivo Python para definir la siguiente función y ejecutarla al final de tu archivo:

- Crear `def print_hello():`

  Esta función no recibe *input* y se encarga de imprimir el siguiente mensaje:

  `"Hola mundo! Este es mi primer print en Avanzada!!!"`

Para comprobar que la función fue correctamente implementada, deberás "llamar" la función al final de tu archivo y después ejecutar el archivo.

Para ejecutar el archivo desde tu terminal, debes ubicarte en la carpeta `Actividades/AC1`y ejecutar la siguiente línea en la terminal:

- `python3 mi_primer_archivo.py`: Ejecuta el código en la terminal, según el flujo del programa realizado.

Si es que el comando `python3` no te funciona, prueba con `py`, `python`, `python3.12` o `py3`.

Finalmente, una vez que hayas completado correctamente el archivo `mi_primer_archivo.py`, súbelo a Github siguiendo las indicaciones de la parte anterior.

Importante

Si hiciste todo lo anterior bien, deberían ver 1 nuevo *commit* en la página de Github y el archivo `mi_primer_archivo.py` dentro de la carpeta `Actividades/AC1`, junto a el código correspondiente.

Entrega

Para finalizar la AC1 y obtener 4 Puntos de Cátedra (PC), debes realizar y subir dos *commits* dentro del plazo indicado. A continuación se explica el contenido de cada *commit*:

- **1er *commit*** Modificar el archivo `Readme.md` ubicado en la carpeta principal de tu repositorio personal.
- **2do *commit*** Crear y completar el archivo `mi_primer_archivo.py` ubicado en la carpeta `Actividades/AC1` de tu repositorio personal.

Si hiciste lo anterior correctamente, podrás ver tus cambios en la página de Github.

Notas

- Recuerda que la ubicación de tu entrega es en **tu repositorio personal**.
- Se recomienda completar la actividad en el orden del enunciado.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo y/o buscarlo en Google.

Sigamos practicando

1. Introduciendo *Type hinting*

A partir de la siguiente sección se mostrarán fragmentos de código que incluirán un elemento llamada *"Type hinting"*. Si ya conoces este concepto, puedes omitir esta sección e ir directamente a la "Enfrentarse a un problema de programación". En otro caso, te recomendamos leer esta sección.

Type hinting es una forma de declarar el tipo esperado de una variable, argumento o atributo por medio de anotaciones en Python. De este modo, ya no es necesario adivinar y/o memorizar el tipo de dato que corresponde a cada elemento, sino que lo dejamos explicito en el código.

A modo de ejemplo:

```
1 def sumar(numero_1, numero_2):
2     return numero_1 + numero_2
```

Sin utilizar *Type hinting*, no sabemos si esta función efectivamente es utilizada para sumar 2 `int`, concatenar 2 `str`, concatenar 2 `list`, etc. No obstante, si tenemos lo siguiente:

```
1 def sumar_type_hinting(numero_1: int, numero_2: int) --> int:
2     return numero_1 + numero_2
```

Con esta nomenclatura se puede entender más rápidamente que `numero_1` y `numero_2` son del tipo `int`, y que la función `sumar_type_hinting` va a retornar un dato del tipo `int`.

Es importante mencionar que *Type hinting* solo es para que el usuario y la interfaz de desarrollo (VSCode por ejemplo) entienda el tipo de dato de cada elemento, pero el código no se va a caer si es que no se respecta lo explicitado. En caso que a `sumar_type_hinting` se le entrega 2 `str` para concatenar, la función se ejecutará exitosamente.

Para información adicional sobre *Type hinting*, recomendamos revisar el contenido *bonus* de la semana 1.

2. Enfrentarse a un problema de programación

Ya habiéndonos familiarizado con el uso de Git y el uso de la terminal, ¡enfrentémonos a un problema de programación!

Para esto, copia los archivos de la carpeta `Actividades/AC1` del Syllabus a tu repositorio personal (misma carpeta) y realiza lo que se indica a continuación.

Introducción

Ayudaremos a una tienda a digitalizar su experiencia de compra, permitiendo a un usuario agregar los productos disponibles a su canasta. Además, introduciremos los archivos de *testing*, que ayudan a corroborar el avance y puntaje de cada actividad.

Ahora, ¡comenzamos con esta desafiante misión!

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- Crear `main.py`: Deberás crear y completar este archivo según lo pedido.
- No modificar `entities.py`: Este archivo de Python contiene algunas clases (OOP) a utilizar en la actividad.
- No modificar `utils/pretty_print.py`: Este archivo de Python contiene funciones necesarias para visualizar la compra.
- No modificar `utils/items.dcc`: Contiene los datos de los ítems en el formato: `'nombre,precio,puntos'`.
- No modificar `tests_publicos`: Carpeta que contiene diferentes `.py` para ir probando si lo desarrollado hasta el momento cumple con lo esperado.

  En la última hoja del enunciado se encuentra un anexo de cómo ejecutar los *tests* por parte o todos.

2.1. Cargar datos

Para que puedas implementar correctamente las funcionalidades, te entregamos las siguientes clases **ya implementadas** en el módulo `entities.py`:

- No modificar `class Item`:

  Clase que representa un producto de la tienda. Incluye sólo el constructor:

  - `def __init__(self, nombre: str, precio: int, puntos: int) -> None`:

    Este es el inicializador de la clase, y asigna los siguientes atributos:

    - `self.nombre` Texto (`str`) que representa el nombre del producto, guarda la información del argumento `nombre`.
    - `self.precio` Entero (`int`) que representa el precio original del producto, guarda la información del argumento `precio`.
    - `self.puntos` Entero (`int`) que representa la cantidad entera de puntos que acumularía un usuario al comprar el producto, guarda la información del argumento `puntos`.

- No modificar `class Usuario`:

  Clase que representa a un usuario, el cual puede adquirir productos en la tienda.

  - `def __init__(self, esta_suscrito: bool) -> None`:

    Este es el inicializador de la clase, y asigna los siguientes atributos:

    - `self.suscripcion` Booleano (`bool`) que representa si el usuario cuenta o no con suscripción, duplica los puntos de cada item agregado a la canasta, guarda la información del argumento `esta_suscrito`.
    - `self.canasta` Lista (`list`) con instancias de `Item`. Inicialmente parte vacía.
    - `self.puntos` Entero (`int`) que representa la cantidad entera de puntos acumulados del usuario. Inicialmente parte con valor 0.

- `def agregar_item(self, item: Item) -> None`:

  Método de clase que agrega un objeto del tipo `Item` a la lista `self.canasta`. Si el usuario cuenta con suscripción, se duplican los puntos del producto.

- `def comprar(self) -> None`:

  Método de encargado de comprar todos los productos de la canasta, traspasando los puntos de cada producto comprado a los puntos del usuario. Finalmente, deja la canasta vacía.

Para cargar los datos y utilizar las clases anteriores. Primero deberás crear un archivo "`main.py`" dentro de la carpeta "`Actividades/AC1`". Utilizando tu herramienta de desarrollo, deberás abrir y editar el archivo Python para lograr el objetivo de esta parte: importar los elementos necesarios de los otros archivos Python y definir las 2 funciones que se detallarán a continuación. Recuerda mantener el PEP8 al escribir el código.

- Crear `def cargar_items() -> list`:

  Esta función debe extraer la información sobre los productos disponibles desde archivo "`items.dcc`", es decir, debe abrir el archivo "`items.dcc`" y por cada línea del archivo generar una instancia de la clase `Item` con la información correspondiente.

  Finalmente, retorna una lista con cada instancias de `Item` que fue generada a partir del archivo leído.

  Por ejemplo, si el archivo es:

  ```
  papas duquesas,1500,10
  palmitos,4000,50
  ```

  Esta función retornará:

  ```
  [Item("papas duquesas", 1500, 10), Item("palmitos", 4000, 50)]
  ```

- Crear `def crear_usuario(tiene_suscripcion: bool) -> Usuario`:

  Esta función recibe el argumento `tiene_suscripcion` y se encarga de crear una instancia de la clase `Usuario`.

  Luego, usando la función `print_usuario` del archivo `pretty_print.py`, se debe imprimir la información de esta instancia.

  Finalmente, la función retorna la instancia del usuario creado.

  Por ejemplo, si esta función se ejecuta como `crear_usuario(True)`, en pantalla se imprimirá:

  `"> Usuario con suscripcion. Puntos: 0"`.

  Luego, la función retornará: `Usuario(True)`.

2.2. Ejecución del programa

Finalmente, para poder visualizar la simulación, debes agregar el siguiente bloque de código al final del archivo `main.py`, y completar lo indicado por las líneas comentadas, con tal de cumplir con lo pedido.

```
if __name__ == "__main__":
    # 1) Crear usuario (puede ser con o sin suscripcion)
    # 2) Cargar los items
    # 3) Imprimir todos los items usando los módulos de pretty_print
    # 4) Agregar todos los items a la canasta del usuario
    # 5) Imprimir la canasta del usuario usando los módulos de pretty_print
    # 6) Generar la compra desde el usuario
    # 7) Imprimir el usuario usando los módulos de pretty_print
```

La primera línea indica que el *script* contenido en el archivo "`main.py`" será ejecutado directamente al ser llamado desde la terminal. Todo el código que esté contenido dentro de esa instrucción condicional será ejecutado linealmente.

Para probar tu código completo de la actividad, puedes ejecutar estas líneas en la terminal dentro del directorio `Actividades/AC1`:

- `python3 main.py`: Ejecuta el código en la terminal, según el flujo del programa realizado.

Si es que el comando `python3` no te funciona, prueba con `py`, `python`, `py3` o `python3.12`.

Es importante mencionar que todas las actividades serán ejecutadas desde el directorio de la misma actividad, es decir, que para ejecutar sus entregas se hará `python3 main.py`, **y no** `python3 AC1/main.py` ni `python3 Actividades/AC1/main.py`.

3. Ejecución de *tests*

En esta actividad se provee de varios archivos `.py` los cuáles contiene diferentes *tests* que ayudan a validar el desarrollo de la actividad. Para ejecutar estos *tests*, **primero debes posicionar tu terminal/consola en la carpeta de la actividad (`Actividades/AC1`)**. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests* de la actividad:

- `python3 -m unittest discover tests_publicos -v -b`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo si escribes lo siguiente en la terminal/consola:

- `python3 -m unittest -v -b tests_publicos.test_funciones`

  Para ejecutar solo el subconjunto de *tests* relacionado a las funciones `cargar_items` y `crear_usuario`.

- `python3 -m unittest -v -b tests_publicos.test_salidas_terminal`

  Para ejecutar solo el subconjunto de *tests* relacionado a la ejecución del `main.py`.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.12`.