

## Relevamiento de conocimientos previos de programación en el nivel universitario

Gonzalo Pablo Fernández<sup>1</sup>, Cecilia Martínez<sup>2</sup>  
y Pablo E. “Fidel” Martínez López<sup>1,3,4</sup>

<sup>1</sup> Universidad Nacional de Quilmes

<sup>2</sup> Universidad Nacional de Córdoba

<sup>3</sup> Centro de Investigación y Desarrollo en Informática Aplicada -  
Universidad Nacional de Hurlingham

<sup>4</sup> Fundación Sadosky

**Resumen** En los últimos años se le ha dado un papel preponderante a la enseñanza de la programación en los niveles obligatorios de la educación. Han surgido también numerosas alternativas para aprender a programar de forma autodidacta. Como resultado de estos procesos, el público que ingresa a la universidad puede tener conocimientos previos de programación heterogéneos. Las carreras universitarias de informática suelen asumir conocimientos nulos sobre programación pero eso está cada vez más alejado de la realidad.

En este trabajo nos proponemos caracterizar, con base en sus conocimientos previos de programación, a la población de estudiantes universitarios que cursan la materia *Introducción a la Programación* en la *Universidad Nacional de Quilmes*, materia que fue pensada originalmente para enseñar conceptos básicos de programación a estudiantes sin ningún conocimiento previo en el área. Desarrollamos un test diagnóstico para indagar experiencias previas en programación, y evaluar algunos conceptos sobre construcciones sintácticas de los lenguajes de programación.

Nuestros hallazgos muestran la enorme heterogeneidad del público, teniendo estudiantes que aprendieron conceptos de programación en distintas etapas (en la escuela, con cursos en línea, de forma autodidacta), con distintos lenguajes de programación (tanto en texto como en bloques) y además, con percepciones muy distintas respecto a cuánto saben realmente.

**Keywords:** Enseñanza de la programación · Nivel universitario · Conocimientos previos

Received May 2024; Accepted June 2024; Published July 2024

<https://doi.org/10.24215/15146774e057>



Esta obra está bajo una Licencia Creative Commons  
Atribución-No Comercial-CompartirIgual 4.0 internacional

ISSN 1514-6774

## Survey of prior programming knowledge at the university level

Gonzalo Pablo Fernández<sup>1</sup>, Cecilia Martínez<sup>2</sup>  
y Pablo E. “Fidel” Martínez López<sup>134</sup>

<sup>1</sup> Universidad Nacional de Quilmes

<sup>2</sup> Universidad Nacional de Córdoba

<sup>3</sup> Centro de Investigación y Desarrollo en Informática Aplicada -  
Universidad Nacional de Hurlingham

<sup>4</sup> Fundación Sadosky

**Abstract.** In recent years, a predominant role has been given to the teaching of programming at all levels of education. Many alternatives have also appeared to learn to program independently. As a result of these processes, the students entering university may have heterogeneous prior programming knowledge. University computer science courses usually assume no prior knowledge about programming but that is increasingly becoming far from reality.

In this work we propose to characterize, based on their previous knowledge of programming, the population of university students who study the subject *Introduction to Programming* at the *National University of Quilmes*, a subject that was originally intended to teach basic programming concepts to students without any prior knowledge in the area. We developed a diagnostic test to investigate previous experiences in programming, and evaluate some concepts about syntactic constructions of programming languages.

Our findings show the enormous heterogeneity of the public, having students who learned programming concepts at different stages (at school, with online courses, self-taught), with different programming languages (both in text and in blocks) and also, with very different perceptions regarding how much they really know.

**Keywords:** Teaching programming · University level · Previous knowledge

## 1. Introducción

La incorporación de contenidos de computación, y sobre todo, programación en niveles cada vez más bajos de escolarización es un fenómeno global, motivado tanto por las nuevas investigaciones sobre Pensamiento Computacional (según las cuales, las habilidades que se aprenden al estudiar Computación son útiles para cualquier persona y no sólo para quienes se van a dedicar profesionalmente a la informática [10], [2]) como por la creciente brecha entre la demanda de profesionales de la informática y la escasa oferta de mano de obra calificada para satisfacerla.

En Argentina en particular, la iniciativa Program.AR dedica grandes esfuerzos por llevar la enseñanza de la programación y otros temas relacionados a las Ciencias de la Computación a las escuelas primarias y secundarias, a través de capacitación docente, generación de contenido y otras actividades relacionadas. Por otro lado se desarrollaron numerosas propuestas de capacitación brindadas por el gobierno nacional en forma de cursos de programación, orientados principalmente a generar mano de obra calificada en poco tiempo. Entre estas propuestas se destaca Argentina Programa por su alcance nacional (más de 80.000 personas, según la página oficial del programa<sup>5</sup>).

Enseñar a programar a estudiantes universitarios requiere abordar simultáneamente un conjunto de saberes complejos [6]. Entre ellos, la sintaxis y la semántica de un lenguaje de programación son dos de los pilares sobre los que se construyen los demás conocimientos. La semántica se refiere a la expresión de las ideas que se intentan realizar a través del lenguaje de programación. La sintaxis incluye las reglas de construcción de las cadenas de símbolos que la computadora procesará para ejecutar la tarea encomendada. Ya en 1976, Ben Shneiderman proponía usar un enfoque espiralado para enseñar a programar distinguiendo claramente el aspecto sintáctico del semántico [9].

En efecto, diversos estudios muestran que el alumnado que aprende programación presenta dificultades en comprender y construir las reglas sintácticas [3], [11]. Xie et al. proponen que las habilidades necesarias para aprender a programar son 4 aunque dos de ellas están relacionadas con la sintaxis (la capacidad para leerla y entenderla y la capacidad para escribirla) y las otras dos están relacionadas con la semántica (la capacidad para identificar construcciones semánticas y la capacidad para usarlas correctamente según la situación) [12].

Respecto al problema de la sintaxis, se ha realizado mucha investigación en torno a cómo superar las barreras iniciales que presentan la sintaxis compleja de los lenguajes de programación, sobre todo en los niveles educativos iniciales. El principal avance gira en torno a los entornos de programación por bloques que, por la forma en que se escriben los programas, impide cometer errores sintácticos. Sin embargo, poco se ha investigado sobre tales cuestiones en el nivel universitario [1]. Ahora que es cada vez más usual que las personas aprendan a programar en los primeros niveles educativos (o incluso, que aprendan a través

---

<sup>5</sup> <https://www.argentina.gob.ar/economia/conocimiento/argentina-programa>

de cursos o de forma autodidacta) se vuelve imprescindible poner el foco en los primeros años de la educación universitaria.

En nuestra práctica como docentes de *Introducción a la Programación* hemos observado a través de los años que los estudiantes tienen dificultades para construir las nociones asociadas a las reglas sintácticas del lenguaje, sobre todo en las referentes a la distinción entre comandos y expresiones. Nos hemos propuesto establecer cuáles son los saberes previos de sintaxis de programación de los estudiantes de *Introducción a la Programación* en el primer año de una carrera universitaria.

En la sección 2 introducimos el marco teórico sobre el que trabajaremos, tanto en cuanto a antecedentes como en cuanto a definiciones. En la sección 3 describimos el estudio realizado, comenzando con la población sobre la que se realizó, siguiendo con el detalle del instrumento administrado y finalizando con el procesamiento realizado sobre los datos recolectados. En la sección 4 analizamos los resultados obtenidos en las distintas preguntas y luego discutimos esos resultados en la sección 5. Finalmente, resumimos nuestras conclusiones en la sección 6.

## 2. Marco Teórico

### 2.1. Antecedentes

Gómez et al. realizan un estudio en el que comparan a dos grupos de estudiantes de primaria. Uno de los grupos había tenido experiencia previa programando en un entorno de programación por bloques y el otro no. Aunque la cantidad de errores sintácticos en ambos grupos fue similar, el grupo que había tenido experiencia previa con bloques tuvo mejores notas en los exámenes, lo que sugiere que la experiencia con bloques les ayudó a desarrollar ciertas habilidades aunque no les facilitó la comprensión de la sintaxis [4].

Joanna Goode realiza una investigación para indagar sobre los saberes tecnológicos de los ingresantes a la Universidad de los Ángeles en función de datos demográficos. Los datos recogidos en encuestas y entrevistas muestran que a mayor acceso y a mejor posición social, mayores son los saberes de tecnología – entre ellos conceptos centrales de programación. Particularmente describe que aquellos estudiantes que pudieron cursar materias de programación en el secundario han podido borrar las diferencias de origen socio-económico y construir saberes digitales de computación [5].

### 2.2. Definiciones Teóricas

Como ya se mencionó en la Introducción, la sintaxis consiste en el conjunto de reglas que establecen la correcta estructura de los símbolos y construcciones que conforman un programa. Cuando hablamos de estructura sintáctica nos referimos al Árbol de Sintaxis Abstracta (AST, por sus siglas en inglés), que es una representación del código en forma de árbol donde cada nodo es una de

las posibles construcciones y sus hijos son las partes de dicha construcción, sin importar qué símbolos las describen en el texto que conforma el programa. Si bien el AST no es parte explícita de los contenidos impartidos por la materia, es una noción que es posible construir a partir de estudiar la estructura sintáctica de cada herramienta del lenguaje y se espera que luego de trabajar con programas los estudiantes perciban esta estructura de forma al menos implícita y que los ayude a comprender mejor la estructura de las ideas que están expresando mediante el programa.

### 3. Metodología

Para responder a nuestra pregunta de investigación sobre cuáles son las nociones previas de AST de los estudiantes de primer año de programación se realizó un test que permite identificar y cuantificar estas nociones. El experimento se llevó a cabo durante la cursada del primer semestre de 2023.

#### 3.1. Población

Nuestros estudiantes asisten a la *Universidad Nacional de Quilmes* (UNQ). Se trata de una universidad pública ubicada a 15 km de la ciudad de Buenos Aires. Hay 3 carreras en la Universidad que tienen como materia obligatoria a *Introducción a la Programación* (InPr): Tecnicatura en Programación Informática (TPI), Licenciatura en Informática (LI) y Licenciatura en Bioinformática (LB). La TPI está casi por completo contenida dentro de la LI y por eso es común que quienes se anotan a la LI se anoten también a la TPI. El 50 % se ha inscripto sólo en la TPI, el 34 % solo en la LI, y el 14 % en ambas. Una estudiante es de la LB.

La materia en la que se realiza el estudio (InPr) es del segundo semestre. Originalmente fue pensada para introducir las bases de la programación y que sea el primer contacto con elementos de programación, utilizando el lenguaje *Gobstones* [8], un lenguaje diseñado específicamente para enseñar a programar. InPr aborda conceptos básicos de programación tales como programas, procedimientos, repetición simple y alternativa condicional, además de sumar parámetros, repetición condicional, funciones, variables y tipos de datos simples (registros y listas), e incorpora nociones de documentación y estilo. Tras un cambio de plan se incorporó una materia previa, *Elementos de Programación y Lógica* (EPyL), como correlativa, pero sin articulación entre los grupos docentes. En esta materia se decidió incluir nuevamente, entre otros temas, los conceptos básicos de programación: programas simples, procedimientos, repetición simple y alternativa condicional, a pesar de que los mismos ya se abordaban en la materia siguiente. Se utiliza el lenguaje QDraw, un lenguaje teórico que fue construido específicamente para esta materia como simplificación de las ideas de Gobstones. Debido a que no hay un número suficiente de computadoras para que los estudiantes puedan realizar sus prácticas, nunca se pensó en realizar una implementación en computadoras de un intérprete o compilador de QDraw que permita utilizarlo

en máquina. Por esas razones, los ejercicios se resuelven en papel y no hay forma automática de validar la corrección de un programa.

La materia InPr se cursa en 7 comisiones de entre 30 y 40 personas cada una. Una de las comisiones es virtual y se dejó fuera del estudio porque el objetivo de esta investigación era analizar los procesos de aprendizaje y se pensó dificultoso de observar estos procesos de manera virtual. Del total de 219 estudiantes que se inscribieron a la materia, participaron del estudio y dieron su consentimiento 157 personas.

### 3.2. Recolección de la información

Para recuperar información sobre las nociones previas de programación se elaboró un test de saberes previos que incluyó una sección inicial de datos demográficos y una segunda parte con las preguntas temáticas. Así, el test consistía en dos grandes momentos.

Una primera serie de preguntas indagaban sobre las condiciones socio-educativas previas del estudiantado (colegio secundario al que asistieron, experiencias previas de programación, asistencia a cursos de programación, etc).

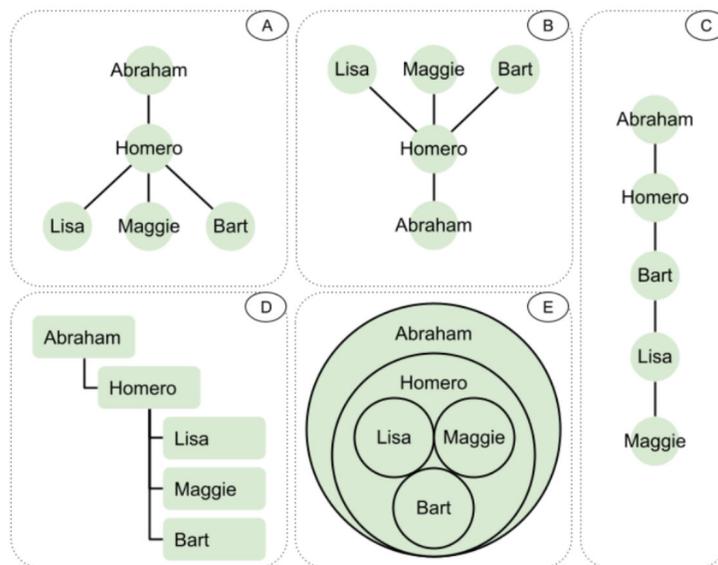
El segundo momento consistía en una serie de actividades que para su resolución requerían poner en juego los conceptos que estamos analizando en este estudio a saber: correcta interpretación de una estructura jerárquica, percepción de estructura sobre determinados conceptos cotidianos, correcta interpretación de la estructura sintáctica de un cuerpo de texto (ya sea código o lenguaje coloquial y la relación entre ambos), clasificación sintáctica de porciones de código en distintos contextos y relación de los mismos con su semántica correspondiente, entre otros.

El instrumento fue administrado el primer día de clase. A continuación se describen cada una de las actividades que incluyó.

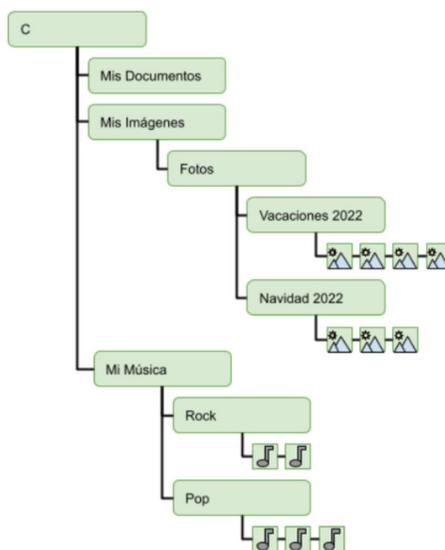
**Árbol Familiar.** La primera actividad se diseñó con el objetivo de validar la interpretación que les estudiantes tienen sobre la estructura jerárquica dada por una representación visual en forma de árbol. Para ello se acudió al concepto de árbol familiar que esperamos que les sea conocido dada su popularidad para expresar las relaciones familiares entre personas. La actividad consistió en dibujar un árbol familiar que expresara la siguiente información.

1. Homero es el padre de Bart.
2. Abraham es el padre de Homero.
3. Lisa es la hija de Homero.
4. Homero es el padre de Maggie.

A cada estudiante se le facilitó una hoja en blanco para que realice su dibujo y solo después de que lo entregara se le permitía continuar con la encuesta. La siguiente consigna ofrecía cinco imágenes y se pedía elegir la que consideraran más adecuada para expresar la información de la consigna anterior y justificar. En la figura 1 se muestran las opciones disponibles. Por último, se pidió que modificaran cada una de las representaciones para agregar que “Herbert es hijo de Abraham”.



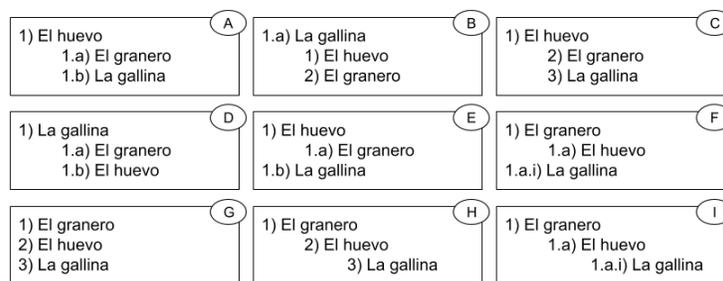
**Figura 1.** Propuestas de posibles representaciones de árboles familiares presentadas para la actividad **Árbol Familiar**.



**Figura 2.** Estructura de directorios para la actividad **Archivos y Carpetas**.

**Archivos y Carpetas.** En la misma línea que la actividad anterior, esta actividad busca validar si pueden comprender correctamente la estructura jerárquica en una situación que esperamos, les sea familiar. En este caso se muestra una estructura de directorios en un sistema de archivos, usando la representación que suelen usar los sistemas operativos con ese fin, como se ve en la figura 2. La consigna pedía determinar la cantidad de fotos de la navidad de 2022 y la cantidad de canciones totales que hay almacenadas en la computadora.

**Huevo y Gallina.** Esta actividad intenta recuperar la noción de indentación que indica la ubicación de acciones de programación en relación a otras. Se presentan varias imágenes que, se supone, representan índices de un libro. Se les pregunta en cuáles la sección “La gallina” es subsección de la sección “El huevo”. Entre las distintas opciones varían el orden de las secciones, la indentación relativa entre las secciones y la numeración de las mismas. Esperamos que quienes no saben programar elijan más aquellas opciones que priorizan la numeración de las secciones por sobre la indentación de las mismas. En la figura 3 se muestran las posibles opciones entre las cuales elegir.



**Figura 3.** Propuestas de posibles índices presentadas para la actividad **Huevo y Gallina**.

**Cantidad o Acción.** El objetivo de esta actividad es recuperar nociones sobre sintaxis y sobre semántica inferidas a partir de fragmentos de código. Se presentan varios fragmentos de código y en cada uno se les pregunta si cierta palabra dentro del código corresponde a una cantidad, a una acción o a otra cosa (en cuyo caso se esperaba que digan a qué otra cosa) y que justifiquen su respuesta. Hay varios indicios (algunos sintácticos y otros semánticos) en cada fragmento que indican una u otra opción y van variando entre los distintos fragmentos. Estos indicios son:

- la categoría sintáctica del nombre de la subtarea (que puede ser un verbo o un sustantivo),
- el significado del nombre de la subtarea (es decir, la semántica del nombre),
- su ubicación dentro del código (que puede ser la ubicación de un argumento o de un cuerpo) y
- el hecho de que inicie o no con una letra mayúscula.

En la figura 4 se muestran cada uno de los fragmentos de código que se pedía clasificar.

<pre> repetir (3) {   comerUnaManzana () } </pre>	<pre> repetir (<u>CantidadDeManzanas</u> ()) {   ComerUnaManzana () } </pre>
<pre> repetir (<u>comerUnaManzana</u> ()) {   cantidadDeManzanas () } </pre>	<pre> repetir (ComerUnaManzana ()) {   <u>CantidadDeManzanas</u> () } </pre>
<pre> repetir (3) {   <u>Manzana</u> () } </pre>	<pre> repetir (<u>contarCantidadDeManzanas</u> ()) {   ComerUnaManzana () } </pre>
<pre> repetir (ContarCantidadDeManzanas ()) {   <u>manzana</u> () } </pre>	<pre> repetir (<u>ContarCantidadDeManzanas</u> ()) {   ComerUnaManzana () } </pre>
<pre> repetir (<u>Manzana</u> ()) {   cantidadDeManzanas () } </pre>	<pre> repetir (ComerUnaManzana ()) {   <u>contarCantidadDeManzanas</u> () } </pre>
<pre> repetir (<u>ComerUnaManzana</u> ()) {   cantidadDeManzanas () } </pre>	<pre> repetir (ComerUnaManzana ()) {   <u>cantidadDeManzanas</u> () } </pre>

**Figura 4.** Fragmentos de código que se pide clasificar en la actividad **Cantidad o Acción**.

Esperamos que quienes no saben programar usen los dos primeros indicadores para determinar si la subtarea en cuestión es una cantidad (es decir, una expresión) o una acción (es decir, un comando), ya que están relacionados al nombre elegido. El uso de mayúsculas no debería influir demasiado porque es un indicador utilizado solamente en la materia: los comandos en Gobstones empiezan con mayúsculas y las expresiones con minúsculas. En cuanto a la ubicación, quienes ya saben programar deberían reconocerlo, sobre todo si conocen lenguajes como C o Java en los que se utiliza la misma sintaxis de cuerpos encerrados entre llaves. Sin embargo, también hay que considerar que, siendo que la sintaxis es casi idéntica a la de QDraw (el lenguaje que ya usaron en la materia correlativa), tiene sentido que presten atención a la estructura de código y sepan que el nombre es solamente un texto sin significado que no tiene ningún efecto en la ejecución. En este experimento se descartaron varios resultados ya que las explicaciones dadas mostraron que no comprendieron la consigna, por lo que el total baja a 141 participantes.

**Cuántas Manzanas.** En esta actividad se busca relevar la forma en que les estudiantes desambiguan frases en lenguaje natural para traducirlas a código. Consta de tres partes. En la primera parte presentan varias oraciones que, de distintas formas, hacen referencia a un número particular describiendo la cantidad de manzanas que se deben comer. Para cada una de ellas se pide calcular cuál es la cantidad total de manzanas que deberían comer si se desea cumplir la orden. Las oraciones que se utilizaron en la primera parte de esta actividad se muestran a continuación.

1. Repetir 3 veces comer una manzana.
2. Repetir 3 veces comer dos manzanas.

3. Repetir 3 veces comer dos manzanas y después comer una manzana más.
4. Mientras haya alguna manzana, comer una manzana.
5. Repetir 3 veces comer una manzana y repetir 2 veces comer dos manzanas.

Las primeras dos se usan como control, ya que no presentan ambigüedad. Tanto en la tercera como en la quinta hay dos interpretaciones posibles. Se puede interpretar que se espera que todo se repita tres veces (“comer dos manzanas y después comer una manzana más” en la tercera y “comer una manzana y repetir 2 veces comer dos manzanas” en la quinta) o que se están pidiendo dos cosas separadas por la “y” y que es la primera (“comer dos manzanas” y “comer una manzana”) la que se pide que se repita tres veces mientras que la segunda (“comer una manzana más” y “repetir 2 veces comer dos manzanas”) debe realizarse aparte, una vez finalizada la repetición. En caso de elegir la primera interpretación diremos que están desambiguando con una *anidación* ya que interpretaron que la segunda parte de la oración está *anidada* dentro de la primera. En el caso contrario, diremos que desambiguaron con una *secuenciación*, ya que interpretaron que la segunda parte debe realizarse a continuación de la primera.

En la segunda parte se pide, no sólo calcular la cantidad total de manzanas sino además determinar cuántas de cada color. Las frases para la segunda parte se muestran a continuación.

1. Repetir 3 veces comer una manzana roja y una verde.
2. Repetir 3 veces comer una manzana roja y después comer una manzana verde.
3. Repetir 3 veces comer una manzana roja y repetir 2 veces comer una manzana verde.
4. Lo que tenés que repetir 3 veces es comer una manzana roja. Comer una manzana verde y comer 2 manzanas rojas.
5. Lo que tenés que repetir 3 veces es comer una manzana roja, comer una manzana verde y comer 2 manzanas rojas.

Nuevamente, la primera es para control. La segunda y la tercera tienen la misma estructura que la tercera y la quinta de la primera parte, donde pueden desambiguar tanto por *anidación* como por *secuenciación*. La cuarta y la quinta son idénticas excepto por un signo de puntuación en el medio de la oración. La cuarta separa las dos partes con un punto mientras que la quinta lo hace con una coma. Estas dos versiones de la misma oración intentan forzar un método de desambiguación (*secuenciación* la primera, al separar claramente las dos partes con un punto y *anidación* la segunda al usar la coma como continuación).

Para la tercera parte se reutilizaron varias de las oraciones de las primeras dos partes pero se modificó la consigna. Ahora, en lugar de responder con números correspondientes a las cantidades de manzanas, se pedía elegir entre porciones de código la que se correspondiera con la oración. A continuación se muestran las frases elegidas para la tercera parte y, para cada una de ellas, las opciones de códigos disponibles para elegir.

1. Repetir 3 veces comer una manzana.

- a) 

```
repetir (3) {  
    comerUnaManzana()  
}
```
  - b) 

```
repetir (3) {}  
comerUnaManzana()
```
  - c) 

```
comerUnaManzana()  
repetir (3) {}
```
2. Repetir 3 veces comer dos manzanas y después comer una manzana más.
- a) 

```
repetir (3) {  
    comerDosManzanas()  
    comerUnaManzana()  
}
```
  - b) 

```
repetir (3) {  
    comerDosManzanas()  
}  
comerUnaManzana()
```
3. Repetir 3 veces comer una manzana y repetir 2 veces comer dos manzanas.
- a) 

```
repetir (3) {  
    comerUnaManzana()  
}  
repetir (2) {  
    comerDosManzanas()  
}
```
  - b) 

```
repetir (3) {  
    comerUnaManzana()  
    repetir (2) {  
        comerDosManzanas()  
    }  
}
```
4. Repetir 3 veces comer una manzana roja y una verde.
- a) 

```
repetir (3) {  
    comerUnaManzanaRoja()  
    comerUnaManzanaVerde()  
}
```
  - b) 

```
repetir (3) {}  
comerUnaManzanaRoja()  
comerUnaManzanaVerde()
```
  - c) 

```
repetir (3) {  
    comerUnaManzanaRoja()  
}  
repetir (3) {  
    comerUnaManzanaVerde()  
}
```
5. Lo que tenés que repetir 3 veces es comer una manzana roja. Comer una manzana verde y comer 2 manzanas rojas.

```

a) repetir (3) {
    comerUnaManzanaRoja()
}
comerUnaManzanaVerde()
comerDosManzanasRojas()
b) repetir (3) {
    comerUnaManzanaRoja()
    comerUnaManzanaVerde()
    comerDosManzanasRojas()
}

```

6. Lo que tenés que repetir 3 veces es comer una manzana roja, comer una manzana verde y comer 2 manzanas rojas.

```

a) repetir (3) {
    comerUnaManzanaRoja()
}
comerUnaManzanaVerde()
comerDosManzanasRojas()
b) repetir (3) {
    comerUnaManzanaRoja()
    comerUnaManzanaVerde()
    comerDosManzanasRojas()
}

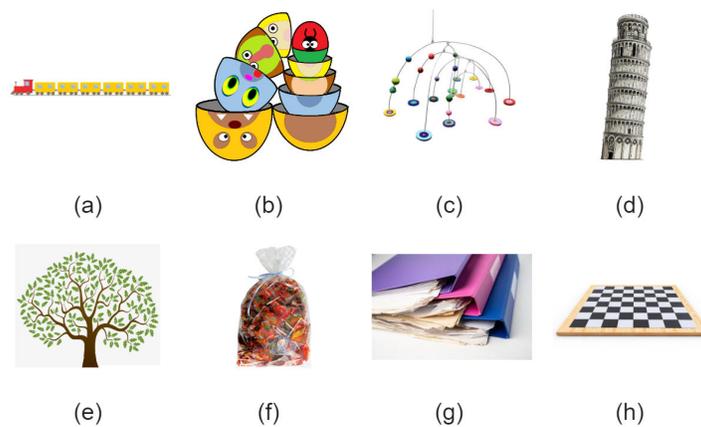
```

La primera y la cuarta son para control. La segunda y la tercera son la tercera y la quinta de la primera parte. La quinta y la sexta son la cuarta y la quinta de la segunda parte.

**Conceptos e imágenes.** En la última actividad se buscó observar en qué medida les estudiantes tienen la capacidad de abstraer la estructura de organización de una colección de elementos. Para ello se presentaron diferentes conceptos cotidianos que consisten a su vez de una colección de elementos y se les pidió, para cada uno de ellos, asociarlo con una imagen de las disponibles. Las imágenes ofrecidas eran siempre las mismas pero se volvían a mostrar en cada pregunta para no generar la idea de que debían hacer una asociación uno a uno entre los conceptos y las imágenes. Cada una de las imágenes, a su vez, expresaba una forma de organización, como un conjunto, una secuencia, un árbol, una matriz, etc. Los conceptos a clasificar fueron “Un árbol genealógico”, “Un diccionario”, “La fila del supermercado”, “Una receta de cocina”, “Una lista de compras”, “Una pirámide”, “Una caja de bombones”, “Un jenga”, “Un candelabro”, “El manual de un electrodoméstico”, “Una secuencia de instrucciones”, “Un álbum de figuritas”, “La nómina de empleados de una empresa”, “El gráfico de la evolución de las especies” y “El sistema solar”. En la Figura 5 se muestran las imágenes disponibles entre las cuales elegir.

### 3.3. Estrategias de análisis

Una primera aproximación a los datos fue a través de estadística descriptiva. Las respuestas del estudiantado se cruzaron en una planilla de cálculo buscando



**Figura 5.** Propuestas de imágenes entre las cuales elegir para clasificar cada concepto en la actividad **Conceptos e imágenes**.

frecuencias y correlaciones. Se identificaron ítems relativos a experiencias previas de aprendizaje de programación con (cursos, materias del secundario, lenguajes, etc).

**Experiencia previa en programación.** Se tabularon las respuestas a la pregunta “¿Sabés programar?” según las categorías “Sí”, “No”, “Poco” (respuestas como “un poco”, “sólo lo básico” o “recién estoy aprendiendo”) y “Sí pero no” (respuestas en las que primero afirman que sí pero luego aclaran que saben HTML y Css). En las preguntas sobre qué lenguajes aprendieron (tanto en el colegio, a través de cursos o de forma autodidacta) se tabularon las respuestas según las categorías “Bloques” (si aprendieron con un entorno de bloques), “Texto” (si aprendieron con un lenguaje textual) y “Web” (si aprendieron HTML y Css). En el caso de los cursos se agregó también la categoría “Argentina Programa” por haber sido mencionada en reiteradas ocasiones.

**Árbol Familiar.** Se clasificaron los dibujos realizados según si cumplían o no la consigna. Dentro de los dibujos correctos se diferenciaron aquellos que ubicaron la raíz del árbol en la parte superior del dibujo y aquellos que lo hicieron en la parte inferior. Luego se asociaron los dibujos con las representaciones elegidas buscando correlaciones.

**Huevo y Gallina.** A cada una de las posibles opciones se les asignó un puntaje positivo o negativo para cada uno de los posibles aspectos (*Numeración e Indentación*) que pudieron haber tenido en cuenta al momento de elegir aquellas opciones que cumplían con la consigna. Así, las opciones que en base a la numeración respetaban la consigna, tuvieron un puntaje positivo para el aspecto *Numeración* mientras que aquellas que en base a la numeración no respetaban la consigna tuvieron un puntaje negativo en tal aspecto. A partir de estos puntajes se calculó el porcentaje de preferencia para cada aspecto como la proporción de opciones elegidas de entre aquellas tales que, según el aspecto,

respetaban la consigna. Si bien la indentación no es una característica exigida por la computadora para que el programa funcione, poner en juego procesos de indentación sería un indicador de que los estudiantes han comprendido la estructura sintáctica del código, de manera que puedan ordenarlo para hacerlo accesible, en primer lugar a ellos mismos en un momento posterior y en segundo lugar a otros programadores.

**Cantidad o Acción.** Se intentó inferir en base a la respuesta qué indicador usaron para decidir. Se tuvieron en cuenta primero los 4 indicadores antes mencionados. El indicador *Posición* establece que si la palabra está entre llaves debe ser un comando y si está entre paréntesis debe ser una expresión. El indicador *Mayúsculas* establece que si la palabra empieza con mayúsculas debe ser un comando y si no una expresión. El indicador *Categoría* establece que si la primera palabra es un verbo debe nombrar un comando y si es un sustantivo debe nombrar una expresión. Para estos primeros 3 indicadores podríamos asociar la elección “Acción” con ser un comando y la elección “Cantidad” con ser una expresión. Sin embargo, según el indicador *Categoría*, las variantes de “Manzana” deberían ser consideradas como “Cantidad” por ser sustantivos. El cuarto indicador es *Semántica* que hace referencia al significado de la oración. Las variantes de “contarCantidadDeManzanas” deberían ser interpretadas como cantidades según este indicador ya que, si bien parece estar describiendo una acción, lo que en definitiva resolvería algo con ese nombre es calcular una cantidad. También hay un detalle con las variantes de “Manzana” ya que teniendo en cuenta el indicador semántico, la respuesta no debería ser ninguna de las dos.

El solapamiento de los indicadores *Categoría* y *Semántica* en las variantes de “Manzana” y “contarCantidadDeManzanas”, nos llevaron a definir un segundo análisis en el que mantuvimos *Posición* y *Mayúsculas* como indicadores y, en lugar de los últimos dos, un tercer indicador basado en el nombre (*Nombre*) que busca incluir ambos aspectos sin ambigüedades. Según este indicador, las variantes de “ComerManzana” serían acciones, las variantes de “cantidadDeManzanas” serían cantidades y las variantes de “Manzana” no serían ninguna de las dos. En el segundo análisis se descartaron las preguntas respecto a las variantes de “contarCantidadDeManzanas”.

Para las opciones en las que la respuesta fue “Otro” se determinó qué indicadores se observaron en base a las justificaciones dadas. Con esta información se calculó cuál fue el indicador preferido por cada participante en base a cuál fue tenido en cuenta en más respuestas que los otros. En caso de empate, consideramos que el participante no tiene preferencia por ninguno.

Finalmente se llevó a cabo un análisis de preferencia definitiva en el que se distinguió a aquellas personas que parecían estar usando siempre el mismo indicador para basar su elección. Decimos que una persona tiene preferencia definitiva por cierto indicador si todas sus respuestas son coherentes con el criterio de tal indicador.

**Cuántas Manzanas.** Las respuestas a la cuarta pregunta de la primera parte (“Mientras haya alguna manzana, comer una manzana”) se tabularon en las siguientes categorías (ya que muchas de ellas fueron en forma de texto y no

de número): “0” (no se come ninguna manzana), “1” (se come una manzana), “1o0” (puede ser una o ninguna pero sin aclarar en base a qué), “1oMas” (puede ser una o pueden ser más pero sin aclarar en base a qué), “IF” (similar a “1o0” pero ahora aclarando la condición de que haya al menos una manzana), “Todo” (se comen todas las que hayan), “Infinito” (la cantidad es infinita), “Indefinido” (la cantidad no se puede saber) e “Inválida” (cualquier otra respuesta que no esté respondiendo la pregunta como “N”, “Depende” o “requisito: debe haber una manzana”). Consideramos correctas las respuestas tabuladas como “Todo”. Las respuestas de la categoría “1oMas” son parcialmente correctas en tanto pudieron detectar la repetición condicional pero no explicaron de qué depende la cantidad y además no consideran la posibilidad de que no haya ninguna.

Para el resto de las preguntas de la primera parte y las de la segunda parte se midió la frecuencia de cada una de las posibles respuestas correctas, distinguiendo cuáles usaron *anidación* y cuáles *secuenciación*. Para las preguntas de la tercera parte también se clasificaron las respuestas en función de si la decisión correspondió a *anidación* o a *secuenciación* y se comparó con las respuestas dadas en las primeras dos partes a las mismas preguntas.

**Conceptos e imágenes.** Se construyó una tabla de doble entrada para contabilizar la cantidad de veces que cada concepto se asoció a cada imagen. Luego se ordenaron las filas y las columnas en función del valor máximo para identificar las asociaciones más fuertes.

## 4. Resultados

A continuación analizamos los resultados obtenidos. Primero observamos las respuestas dadas por los participantes en el primer momento del estudio, donde se les preguntó por su contexto socio-económico así como sobre sus conocimientos previos en programación. Luego estudiamos las respuestas para las actividades del segundo momento, a partir de las cuales podemos inferir sus intuiciones respecto a varios conceptos relacionados a la tarea de programar.

### 4.1. Contexto previo

Un primer hallazgo es que el 50% del estudiantado cursó materias de programación en el nivel secundario. Respecto de la formación previa de nuestros estudiantes en programación se pudo reconstruir a partir de la encuesta que el 20% fue a un colegio técnico o con orientación en informática. El 64% no fue a colegio técnico o con orientación en informática pero igual tuvo alguna materia relacionada (informática, computación, tics, etc.) en su escuela secundaria.

En Argentina, las escuelas técnicas y con orientación en informática ofrecen como materias obligatorias tres materias de Programación, Aplicación a las Nuevas Tecnologías (que incluye base de datos, normas de seguridad, etc.), y Formación en Ambientes del Trabajo. En tanto en la formación general básica informática incluye generalmente habilidades que permiten usar programas, o tecnologías de la información y la comunicación. Entre ellas se incluyen el uso

de procesadores de textos o planillas de cálculos. Sin embargo, al preguntarles si habían aprendido algún lenguaje de programación (ya sea de texto o de bloques) en la escuela, menos del 15% respondió de forma afirmativa. Sí observamos una mayor cantidad de gente que aprendió a programar a través de cursos o de forma autodidacta. Un 10% del estudiantado encuestado mencionó haber realizado (o estar realizando en el momento) el curso Argentina Programa.

A pesar de haber aprobado EPyL, más de la mitad (55%) aseguró no saber programar. El 30% aseguró que sí sabe aunque al preguntarles qué lenguajes conocían mencionaron lenguajes que no son de programación (como html o css). Otro 10% dijo saber pero destacó que sabe muy poco o sólo lo básico. Sólo 4 mencionaron QDraw.

#### 4.2. Percepción de la estructura de organización

Los resultados de las actividades del árbol familiar y del sistema de archivos sugieren que la gran mayoría comprende la idea de estructura jerárquica. El 84% de los árboles dibujados cumplía la consigna. El formato más elegido (71% del total) fue ubicando la raíz en la parte superior del dibujo. Sólo 4 (menos del 3%) lo realizaron de forma invertida. De los 25 que no cumplían la consigna, en 15 la razón por la que no lo hicieron fue que estaban incompletos.

Al presentarse las 5 posibles representaciones, las más elegidas fueron la A (77%) y la D (13%). Las 4 personas que habían dibujado el árbol con la raíz abajo eligieron la opción B. De las 111 que habían dibujado el árbol con la raíz arriba, sólo 88 eligieron la opción A. Del resto, 14 eligieron la D, 7 la B, 1 la C y 1 la E.

Las respuestas a las preguntas sobre el sistema de archivos también fueron en su mayoría correctas. De las 157 personas sólo 4 respondieron de forma incorrecta a la cantidad de fotos y sólo 1 a la cantidad de canciones.

La tabla de la figura 6 muestra en qué porcentajes fue asociado cada concepto con cada imagen. Para los primeros 8 conceptos hay una imagen que se lleva más del 50% de las asociaciones. El concepto “Secuencia de instrucciones” fue asociado principalmente con la imagen del tren (42%). Sin embargo, también hubo un importante porcentaje de participantes (20%) que en su lugar lo asociaron con la imagen de la mamushka.

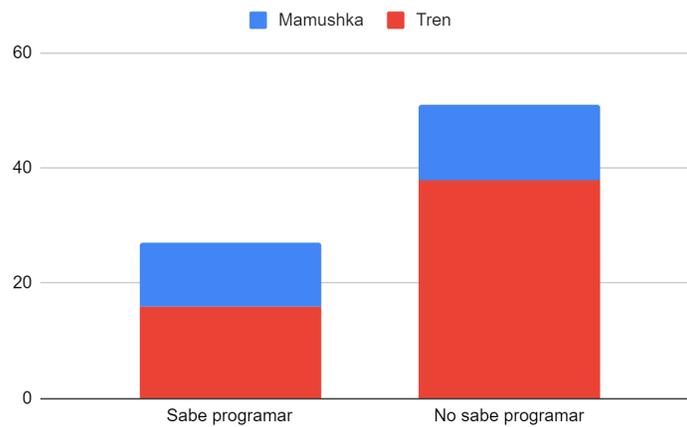
La figura 7 compara las asociaciones del concepto “Secuencia de instrucciones” con cada una de estas imágenes y cruza esos datos con las respuestas a la pregunta “¿Sabés programar?”. Aunque parece haber una mayor tendencia a elegir el tren entre la gente que dijo no saber programar, no parece haber una correlación entre saber programar y asociar una secuencia de instrucciones más a una mamushka (una estructura que se anida) que a un tren (una estructura secuencial).

#### 4.3. Indentación

Respecto a la noción de indentación observamos que la mayor parte del estudiantado le dio mayor peso al aspecto semántico que al sintáctico cuando

	Tren	Colgante	Carpeta	Torre	Árbol	Bolsa	Mamushka	Tablero
La fila del supermercado	95%	0%	1%	1%	0%	1%	1%	1%
El sistema solar	4%	89%	2%	1%	1%	0%	3%	0%
El manual de un electrodoméstico	6%	4%	83%	1%	1%	1%	3%	2%
Un candelabro	2%	78%	3%	3%	12%	0%	0%	3%
Un diccionario	6%	3%	77%	4%	2%	6%	2%	1%
Un jenga	0%	3%	1%	76%	0%	1%	7%	12%
Un árbol genealógico	0%	16%	1%	0%	69%	0%	13%	1%
Una caja de bombones	1%	1%	1%	0%	1%	60%	13%	24%
La nómina de empleados de una empresa	6%	11%	43%	4%	9%	6%	13%	8%
Una pirámide	1%	14%	2%	47%	3%	3%	25%	4%
Un álbum de figuritas	1%	1%	46%	0%	4%	10%	3%	34%
Una secuencia de instrucciones	42%	9%	15%	4%	1%	0%	20%	10%
El gráfico de la evolución de las especies	8%	20%	3%	1%	41%	0%	28%	0%
Una lista de compras	27%	4%	38%	11%	1%	15%	3%	3%
Una receta de cocina	13%	8%	35%	4%	1%	8%	25%	4%

**Figura 6.** Porcentaje de asociación a cada imagen por cada concepto. Tanto las filas como las columnas están ordenadas de forma decreciente.

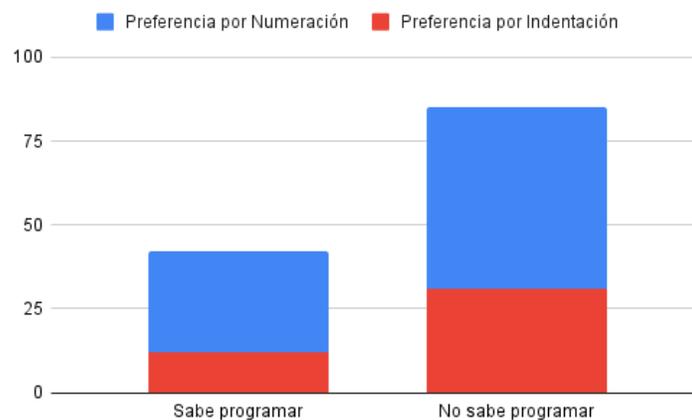


**Figura 7.** Cantidad de personas que asociaron el concepto “Secuencia de instrucciones” con cada imagen, agrupadas según si afirmaron o negaron saber programar.

tuvieron que determinar cuál era la jerarquía implicada en la actividad del huevo y la gallina. Al considerar el porcentaje de estudiantes que eligió en base a la numeración o a la indentación, la relación fue 70% a favor de la numeración contra 30% de la indentación.

En una segunda revisión nos enfocamos en las opciones que favorecían Numeración pero contradecían Indentación (las opciones E y F) y las que hacían lo contrario (C y H). Comparamos entonces cuántas personas eligieron E y F pero no C y H y cuántas eligieron C y H pero no E y F. Observamos en este caso que más del 30% eligió las dos de numeración y no las dos de indentación y sólo un 5% hizo la elección inversa: eligió las dos de indentación y no las dos de numeración.

Al cruzar los datos de preferencia con las respuestas a la pregunta de si sabían programar vemos que no parece haber correlación. Se tuvieron en cuenta para esto sólo las respuestas que fueron afirmativas o negativas (ignorando las dudosas). Como se puede observar en la figura 8, pareciera que la numeración sigue siendo el aspecto más elegido sin importar el conocimiento previo en programación.

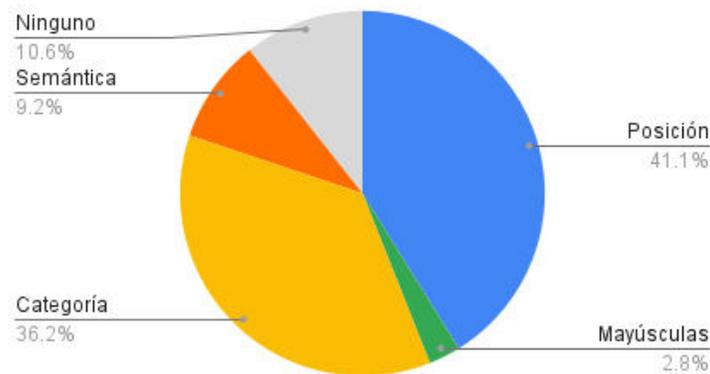


**Figura 8.** Cantidad de personas que manifestaron preferencia por alguno de los indicadores *Numeración* o *Indentación*, agrupadas según si afirmaron o negaron saber programar.

#### 4.4. Clasificación

Respecto a la noción de clasificación de fragmentos, en ambos análisis se observó que el indicador más elegido fue *Posición*. En el primer análisis con 4 indicadores, el segundo indicador más elegido fue *Categoría*. En el segundo análisis con 3 indicadores, el indicador *Nombre* fue casi tan elegido como el indicador *Posición*. En ambos casos el indicador *Mayúsculas* fue el menos preferido. Los

porcentajes de preferencia por cada indicador se pueden apreciar en las figuras 9 y 10 respectivamente. En el caso de la preferencia definitiva, observamos que



**Figura 9.** Porcentaje de participantes que manifestaron preferencia por cada uno de los cuatro indicadores en el primer análisis.

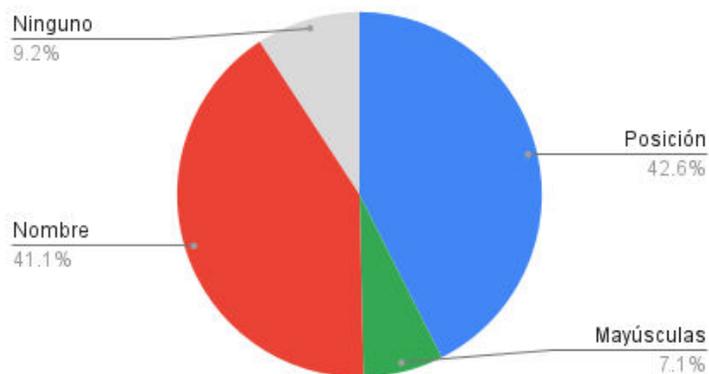
25 % usó siempre la posición para elegir, 10 % eligió siempre en base al nombre y menos de 1 % (una única persona) eligió siempre en base a las mayúsculas. Esperando encontrar alguna correlación entre conocimientos previos en programación y la predilección por el indicador *Posición*, cruzamos estos datos con las respuestas a la pregunta de si sabían programar. Nuevamente, se tuvieron en cuenta para esto sólo las respuestas que fueron afirmativas o negativas (ignorando las dudosas). Este análisis se puede apreciar en la figura 11.

#### 4.5. Desambiguación del lenguaje natural

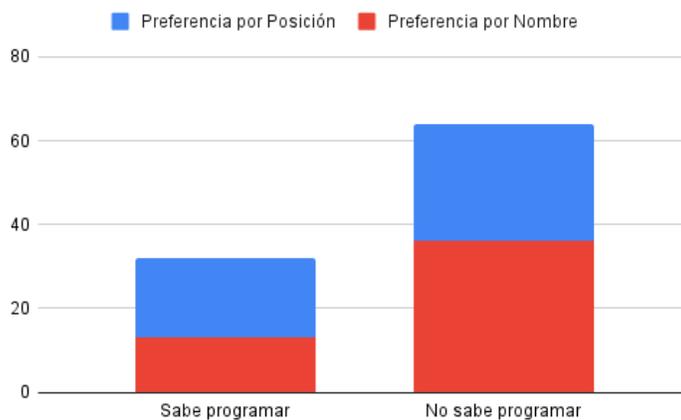
Al desambiguar la frase “Mientras haya alguna manzana, comer una manzana” hubo muchas respuestas que parecen haber interpretado una alternativa condicional (*si hay manzana*) en lugar de una repetición condicional (*mientras haya manzana*). De las 132 respuestas válidas 20 fueron tabuladas como “IF” (15 %) y otras 7 como “1o0” (5 %). Sin embargo, la respuesta más frecuente fue “1” (36 %). Las respuestas correctas (tabuladas como “Todo”) fueron 33 (25 %).

Las respuestas a las preguntas de control fueron mayormente correctas, aunque aparecieron varias incorrectas. Para la frase “Repetir 3 veces comer una manzana”, 151 de las 157 respuestas fueron correctas. Para la frase “Repetir 3 veces comer dos manzanas” hubo 147 respuestas correctas. Entre las incorrectas aparecieron “2”, “3”, “7”, “8” y “9”. En el caso de “Repetir 3 veces comer una manzana roja y una verde”, si bien la respuesta más frecuente fue “3 y 3” (con 138 apariciones) también se encontraron 13 que respondieron “3 y 1”.

Cuando tuvieron que asociar las frases “Repetir 3 veces comer una manzana” y “Repetir 3 veces comer una manzana roja y una verde” con porciones de código,



**Figura 10.** Porcentaje de participantes que manifestaron preferencia por cada uno de los tres indicadores en el segundo análisis.



**Figura 11.** Cantidad de personas que manifestaron preferencia por alguno de los indicadores *Posición* o *Nombre*, agrupadas según si afirmaron o negaron saber programar.

la proporción de respuestas correctas fue aún mayor: 154 y 152 respectivamente. De las 13 personas que respondieron “3 y 1” a la frase “Repetir 3 veces comer una manzana roja y una verde”, 12 eligieron el código que repite 3 veces comer una manzana de cada una. Sin embargo, como esta frase había sido pensada como control, no se había considerado la posibilidad de que interpretaran una *secuenciación* en lugar de una *anidación* y, por lo tanto, no se había planteado una posible opción que se correspondiera a tal interpretación.

Para las frases ambiguas, la mayoría de las respuestas fueron interpretando *secuenciación* en lugar de *anidación*. Las respuestas a “Repetir 3 veces comer dos manzanas y después comer una manzana más” fueron 115 interpretando *secuenciación* y 31 interpretando *anidación*, mientras que las respuestas a “Repetir 3 veces comer una manzana y repetir 2 veces comer dos manzanas” fueron 124 y 12, respectivamente. En total 102 personas interpretaron en ambas *secuenciación* y solo 7 interpretaron en ambas *anidación*.

Al tener que seleccionar una porción de código para estas frases, la mayoría de las respuestas son consistentes. Las interpretaciones de *secuenciación* fueron 145 y 131 respectivamente. Para la primera hubo 113 participantes que interpretaron *secuenciación* tanto al responder la cantidad como al asociar el código. Para la segunda esta cantidad fue 104. En ambos casos hubo participantes que interpretaron *secuenciación* al responder pero *anidación* al asociar (2 en la primera y 20 en la segunda) y viceversa (22 en la primera y 7 en la segunda).

Los resultados en la segunda parte son similares. Las interpretaciones de *secuenciación* y *anidación* para la frase “Repetir 3 veces comer una manzana roja y repetir 2 veces comer una manzana verde” fueron 125 y 20 respectivamente, mientras que para la frase “Repetir 3 veces comer una manzana roja y después comer una manzana verde” fueron 118 y 32. La cantidad de participantes que interpretaron en ambas *secuenciación* fue 101 y la de participantes que interpretaron en ambas *anidación* fue 9.

Recién en las últimas frases de la segunda parte, pensadas específicamente para distinguir las dos interpretaciones, empieza a ganar más peso la interpretación de *anidación*. Para la frase “Lo que tenés que repetir 3 veces es comer una manzana roja. Comer una manzana verde y comer 2 manzanas rojas” (la que se esperaba que interpreten *secuenciación* por tener un punto y aparte) las interpretaciones fueron 128 *secuenciación* y 11 *anidación*. Para la frase “Lo que tenés que repetir 3 veces es comer una manzana roja, comer una manzana verde y comer 2 manzanas rojas” (la que se esperaba que interpreten *anidación* por tener una coma) las interpretaciones fueron 51 *secuenciación* y 83 *anidación*.

Esta última oración es la única en la que hubo más respuestas interpretando *anidación* que *secuenciación* aunque la diferencia no es tan grande como se esperaba. Para estas últimas, la cantidad de participantes que interpretaron *secuenciación* en ambas fue 48 y que interpretaron *anidación* en ambas fue 11. Hubo 69 participantes que realizaron la interpretación esperada (*secuenciación* en la primera y *anidación* en la segunda).

El mismo efecto se percibe en las asociaciones de estas frases a porciones de código. En la primera, las interpretaciones de *secuenciación* fueron 147 mientras

que en la segunda fueron solo 29. La primera tuvo mayor consistencia al comparar las respuestas dadas con las porciones de código elegidas. Hubo 122 participantes que interpretaron *secuenciación* tanto al responder la cantidad como al elegir la porción de código. En el caso de la segunda, hubo 77 que interpretaron *anidación* en ambas instancias y 33 que al responder interpretaron *secuenciación* pero al elegir una porción de código interpretaron en su lugar *anidación*.

Hubo 23 participantes que seleccionaron una porción de código interpretando *secuenciación* en las frases no control y 93 que interpretaron *secuenciación* en las primeras 3 y *anidación* en la cuarta.

## 5. Discusión

### 5.1. Contexto previo

Aunque la mitad de los estudiantes que participaron en el estudio dijeron haber cursado alguna materia relacionada a la Computación, menos del 15% mencionó haber aprendido a programar en la escuela. Esto nos dice que aún estamos lejos de poder asumir conocimientos básicos de programación por parte de estudiantes que terminan la escuela media.

La formación recibida en la materia EPyL tampoco parece ser confiable, puesto que más de la mitad aseguró no saber programar y sólo 4 de las 157 personas que participaron mencionaron el lenguaje de programación utilizado en la materia.

Podemos observar que los antecedentes formales académicos no son un indicador válido de los conocimientos previos, pues a pesar de haber tenido materias de programación en la secundaria y de haber cursado una materia previa con contenidos de programación, los estudiantes manifiestan no saber esos conocimientos.

Asimismo, otra observación es que pareciera que los estudiantes no identifican las nociones previas a poder programar como parte de saberes de programación. Ellos mencionan que “no saben programar” a pesar de haber logrado en las experiencias previas construcciones sencillas donde se usan los principales conceptos. Dos hipótesis nos permitirían explicar este fenómeno.

Por un lado la confianza – o falta de confianza – en sus posibilidades de programar. Esto se evidencia en la baja proporción de estudiantes que aseguraron saber programar (con un rotundo “no” como respuesta) o que dieron respuestas tímidas como “sí, pero poco” o “recién estoy empezando”.

Por otro lado, la idea de que programar construcciones sencillas en Scratch u otros entornos didácticos “no es programar”. Esto se evidencia por un lado en el hecho de que sólo 4 personas hayan mencionado el lenguaje QDraw en la respuesta a los lenguajes de programación conocidos y por otro en la aparente nula relación entre haber usado entornos de programación por bloques y saber programar. De las 76 personas que mencionaron haber utilizado algún entorno de programación por bloques, 36 aseguraron no saber programar (casi la mitad) y sólo 20 (27%) aseguraron que sí, aunque todos conocían además algún lenguaje textual industrial.

Esta cuestión fue observada por Colleen Lewis al comparar dos grupos de estudiantes aprendiendo a programar. Uno de los grupos aprendió con Logo y el otro con Scratch. Al indagar las percepciones de los estudiantes al finalizar el curso quienes habían aprendido a programar con Logo (un lenguaje basado en texto) aparentaban tener mayor confianza en sus habilidades que quienes aprendieron con Scratch (un lenguaje basado en bloques) [7].

## 5.2. Percepción de la estructura de organización

Las primeras actividades del segundo momento muestran que, en general, perciben correctamente la jerarquía dada por una estructura en forma de árbol. En la actividad de asociar conceptos con imágenes, la mayoría de las asociaciones parece haber sido motivada por la estructura, aunque hay algunas excepciones. Los conceptos “El manual de un electrodoméstico”, “Un diccionario”, “La nómina de empleados de una empresa”, “Un álbum de figuritas”, “Una lista de compras” y “Una receta de cocina” tuvieron un alto porcentaje de asociación con la imagen de la carpeta. Creemos que en estos casos, la asociación pudo haber sido basada más en el material (todos los conceptos remiten a algo hecho en papel) que a la estructura de organización. Es posible que la imagen elegida no haya sido adecuada ya que en ella sobresalen los papeles dentro de las carpetas y no tanto la carpeta en sí.

Otro defecto detectado en esta actividad fue el hecho de haber utilizado el concepto “Secuencia de instrucciones” en lugar de “Programa” ya que el uso de la palabra “Secuencia” es demasiado sugerente. Esto pudo haber provocado mayor preferencia por la imagen del tren, ignorando la posibilidad de que las partes de un programa puedan estructurarse de otra forma que no sea secuencial.

## 5.3. Indentación

Más allá de si respondieron que sabían o que no sabían programar, la mayoría de los estudiantes parecen inclinarse a favorecer la numeración por sobre la indentación al inducir la estructura jerárquica de los elementos en cuestión. Estos datos nos estarían sugiriendo que la noción de jerarquía sintáctica por indentación no parece estar interiorizada como un elemento a tener en cuenta para determinar la jerarquía de la estructura.

Sin embargo, también creemos que el enunciado de la actividad podría haber sido demasiado sugerente y haber influido en las decisiones tomadas. Al hablar el enunciado sobre “las secciones de un índice” es posible que hayan priorizado la numeración por el simple hecho de que es lo que se suele usar para marcar jerarquía en ese contexto. Quizás realizando una actividad similar pero en un contexto de programación o en otro contexto más “neutral” los resultados serían diferentes.

## 5.4. Clasificación

La elección del indicador *Posición* por sobre el indicador *Nombre* al clasificar porciones de código en acciones o cantidades nos hace pensar que estas personas

conocen el funcionamiento de los lenguajes de programación en los que los nombres son ignorados por la computadora cuando el código se ejecuta mientras que las personas que eligieron *Nombre* por sobre *Posición* podrían no saber cómo es la estructura sintáctica de los lenguajes de programación y se guiaron por lo que sí entendían en el fragmento presentado. Sin embargo también podríamos pensar que este segundo grupo eligió en base al nombre porque esperan que el código use nombres declarativos.

Al cruzar estos datos con las respuestas a la pregunta de si saben programar, observamos que no parece haber correlación entre autopercepción que saben programar desde antes y haber tenido más en cuenta la posición (es decir la sintaxis) que el nombre (es decir la semántica). Como en el caso anterior, esto podría explicarse si suponemos que quienes saben programar esperan que el código use nombres declarativos y por eso hay una gran proporción que le da prioridad al nombre, aunque creemos que la explicación puede ser en realidad que la respuesta a la pregunta de si saben programar no haya sido del todo honesta. Volviendo a recordar que todos los participantes cursaron la materia EPyL, quienes negaron saber programar podrían haber interiorizado ciertos conceptos que ahora están poniendo en juego. En otras palabras, sí aprendieron a programar (o al menos, aprendieron algunos conceptos) pero no perciben los saberes adquiridos como habilidades relacionadas a la programación.

Los datos sobre preferencia definitiva nos muestran que la gran mayoría no tiene un criterio claro para decidir cuando algunos indicadores contradicen a otros. También vemos que la proporción que elige en base a la posición ignorando totalmente los demás indicadores es bastante mayor a la que tiene esa preferencia con el nombre, a pesar de que en el análisis anterior estos dos grupos parecían ser de similar dimensión.

### 5.5. Desambiguación del lenguaje natural

Notamos una importante cantidad de estudiantes que interpretaron la oración “Mientras haya alguna manzana, comer una manzana” como una alternativa condicional. Esto podría sugerir que adquirieron correctamente el concepto de alternativa condicional durante la cursada en EPyL y creyeron estar en presencia del mismo. Sin embargo, también podría interpretarse que la adquisición de tal concepto no fue del todo efectiva y por eso no pudieron identificarla correctamente.

Al analizar la preferencia por *anidación* o *secuenciación* al desambiguar, la segunda parece ser mucho más frecuente que la primera. Esto podría ser un indicador de la dificultad para interpretar la estructura en forma de árbol. Si bien las primeras actividades mostraron que pueden entender la estructura al organizar elementos simples como personas o archivos, es posible que esa misma facilidad no se transfiera al contexto en el que deben organizar partes de una oración de la misma forma.

Sí observamos una importante cantidad de participantes que pudo distinguir las últimas dos oraciones y clasificarlas correctamente en función de si las dos partes estaban unidas por un punto o por una coma.

## 6. Conclusiones

Se realizó un relevamiento de los conocimientos previos en programación a estudiantes que cursan la materia Introducción a la Programación en la Universidad Nacional de Quilmes. Aunque la materia fue pensada asumiendo nulos (o muy escasos) conocimientos previos en programación, como docentes empezamos a notar cada vez más que no es esa la situación. El objetivo de este trabajo de investigación no es en este momento plantear propuestas para realizar modificaciones en el dictado de la materia sino comprender el contexto inicial para luego avanzar en las modificaciones que sean pertinentes.

Un primer hallazgo es que el grupo es muy heterogéneo en cuanto a conocimientos previos de programación. Teniendo en cuenta únicamente la respuesta a la pregunta “¿Sabés programar?” ya observamos una gran disparidad. Sin embargo, en un análisis más detallado de las respuestas, observamos que no es suficiente con separar al curso entre quienes saben y quienes no saben programar ya que encontramos una gran cantidad de puntos intermedios (gente que dice que no sabe programar a pesar de que utilizó algún entorno de programación por bloques, gente que dice que no sabe programar pero identifica correctamente los elementos de programación, etc).

Recordando que todos los participantes tuvieron que cursar la materia correlativa EPyL, observamos que los contenidos impartidos no alcanzan para que los estudiantes perciban que saben programar, aunque hay varios indicios que nos muestran que sí parecen haber adquirido varios saberes relacionados a la disciplina. Esto puede deberse a que no está clara la definición de programar y, por lo tanto, no consideren que todo lo que aprendieron en EPyL sea programar.

En cuanto a los saberes previos en programación, saber programar (o creerlo) no parece ser un factor determinante al momento de realizar las actividades. Independientemente de si sabían o no programar, la numeración de las secciones fue el aspecto más elegido para decidir la jerarquía entre ellas. En el caso de la clasificación en comandos y expresiones, se ven similares proporciones de personas que deciden en base a la posición y que deciden en base al nombre. Incluso entre quienes afirmaron saber programar, hay una importante proporción que presta atención al nombre, a pesar de que este no sea determinante para la correcta ejecución del código. Tampoco se aprecian diferencias significativas entre estos grupos en cuanto a su percepción de la estructura jerárquica del código. Nos surge sin embargo la incógnita de si las actividades planteadas son efectivas para evaluar conocimientos en programación, o al menos las nociones que intentamos relevar.

Queda como trabajo futuro reunir al plantel docente de la materia para replantear ciertos aspectos de su dictado teniendo en cuenta los resultados obtenidos así como también articular mejor la transición desde la materia correlativa EPyL. Este problema se abordará en otra etapa del trabajo.

También se espera volver a realizar el estudio en futuros dictados de la materia para, por un lado, realizar las correcciones mencionadas a las preguntas (como reemplazar la imagen de la carpeta o cambiar el concepto “Secuencia de

instrucciones” por “Programa”) y por otro, estudiar si la situación se mantiene o se va modificando.

## Referencias

1. Abdul-Rahman, Siti-Soraya & du Boulay, Benedict: Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, 30, 286–298 (2014).
2. Denning, Peter J. & Tedre, Matti: *Computational Thinking*. The MIT Press Essential Knowledge Series. MIT Press. ISBN: 9780262536561 (2019).
3. Dijkstra, Edsger W.: On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12), 1398–1404 (1989).
4. Gomez, Marcos J. & Moresi, Marcos & Benotti, Luciana: Text-based Programming in Elementary School: A Comparative Study of Programming Abilities in Children with and without Block-based Experience. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 402–408 (2019).
5. Goode, Joanna: Mind the Gap: The Digital Dimension of College Access. *The Journal of Higher Education*, 81(5), 583–618. <http://www.jstor.org/stable/40835720> (2010).
6. Jenkins, Tony: On the Difficulty of Learning to Program. 3rd Annual LTSN-ICS Conference, Loughborough University (2002).
7. Lewis, Colleen M.: How programming environment shapes perception, learning and goals: Logo vs. Scratch. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 346–350 (2010).
8. Martínez López, Pablo E. & Bonelli, Eduardo A. & Sawady O’Connor, Federico A.: El nombre verdadero de la programación. Una concepción de enseñanza de la programación para la sociedad de la información. *Anales del 10o Simposio de la Sociedad de la Información (SSI’12)*, dentro de las 41 Jornadas Argentinas de Informática e Investigación Operativa (JAIIO’12). Páginas 1–23, Universidad Nacional de La Plata (UNLP), La Plata, Argentina. ISSN: 1850-2830 (2012).
9. Shneiderman, Ben. Teaching programming: A spiral approach to syntax and semantics. *Computers & Education*, 1(4), 193–197 (1977).
10. Wing, Jeannette M.: Computational Thinking. *Communications of the ACM*, 49(3), 33-35 (2006).
11. Winslow, Leon. E.: Programming pedagogy - a psychological overview. *SIGCSE Bulletin*, 28(3), 17–22 (1996).
12. Xie, Benjamin & Loksa, Dastyni & Nelson, Greg L. & Davidson, Matthew J. & Dong, Dongsheng & Kwik, Harrison & Hui Tan, Alex & Hwa, Leanne & Li, Min & Ko, Amy J.: A theory of instruction for introductory programming skills. *Computer Science Education*, 29(2), 205–253 (2019).