Relevamiento de conocimientos previos de programación en el nivel universitario

Gonzalo Pablo Fernández[1], Cecilia Martínez[2], and Pablo E. "Fidel" Martínez López[1]

[1] Universidad Nacional de Quilmes
[2] Universidad Nacional de Córdoba

Abstract. 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 relevar 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, así como para evaluar algunos conceptos respecto a 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

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.

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 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 les estudiantes tienen dificultades para con-

struir 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 les 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. Finalmente, resumimos nuestras conclusiones en la sección 5.

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 aquelles 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), 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 les 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 les 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. Un 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 decidieron 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 construído específicamente para esta materia como simplificación de las ideas de Gobstones. Debido a que no hay un número suficiente de computadoras para que les 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. El test ofreció 7 actividades, de las cuales seleccionamos 2 de ellas por ser las más relevantes. Las actividades elegidas se detallan a continuación.

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 1 se muestran las posibles opciones entre las cuales elegir.

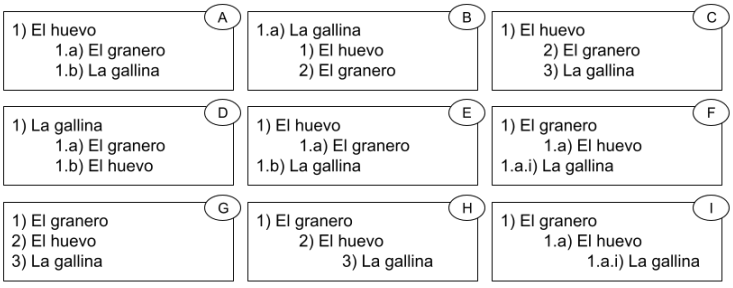

Fig. 1. Propuestas de posibles índices presentadas para la pregunta **Huevo y Gallina**.

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 2 se muestran cada uno de los fragmentos de código que se pedía clasificar.

repetir (3) {
comerUnaManzana()
}
	repetir (CantidadDeManzanas()) {
	ComerUnaManzana()
	}

repetir (comerUnaManzana()) {
cantidadDeManzanas()
}
	repetir (ComerUnaManzana()) {
	CantidadDeManzanas()
	}

repetir (3) {
Manzana()
}
	repetir (contarCantidadDeManzanas()) {
	ComerUnaManzana()
	}

repetir (ContarCantidadDeManzanas()) {
manzana()
}
	repetir (ContarCantidadDeManzanas()) {
	ComerUnaManzana()
	}

repetir (Manzana()) {
cantidadDeManzanas()
}
	repetir (ComerUnaManzana()) {
	contarCantidadDeManzanas()
	}

repetir (ComerUnaManzana()) {
cantidadDeManzanas()
}
	repetir (ComerUnaManzana()) {
	cantidadDeManzanas()
	}

Fig. 2. Fragmentos de código que se pide clasificar en la pregunta **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.

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

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 les estudiantes han comprendido la estructura sintáctica del código, de manera que puedan ordenarlo para hacerlo accesible, en primer lugar a elles mismes en un momento posterior y en segundo lugar a otres programadoros.

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 le 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.

4 Resultados

A continuación analizamos los resultados obtenidos. Primero observamos las respuestas dadas por les 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 dos actividades del segundo momento, a partir de las cuales podemos inferir sus intuiciones respecto a la indentación del código y a la correcta identificación de elementos en un fragmento de código, respectivamente.

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.

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, les estudiantes manifiestan no saber esos conocimientos.

Asimismo, otra observación es que pareciera que les estudiantes no identifican las nociones previas a poder programar como parte de saberes de programación. Elles 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 todes 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 les 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].

4.2 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 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.

Estos datos nos permiten deducir 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.

Al cruzar los datos de preferencia con las respuestas a la pregunta de si sabían programar vemos que no parece haber correlación. Como se puede observar en la figura 3, pareciera que la numeración sigue siendo el aspecto más elegido sin importar el conocimiento previo en programación.

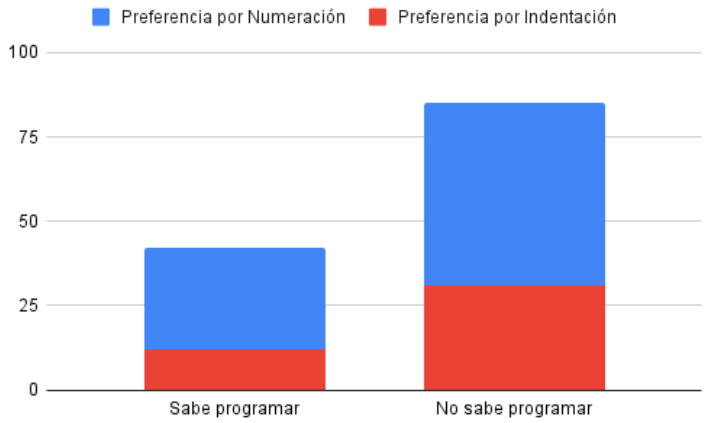

Fig. 3. Cantidad de personas que manifestaron preferencia por alguno de los indicadores *Numeración* o *Indentación*, agrupadas según si afirmaron o negaron saber programar.

4.3 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 4 y 5 respectivamente.

La elección del indicador *Posición* por sobre el indicador *Nombre* 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

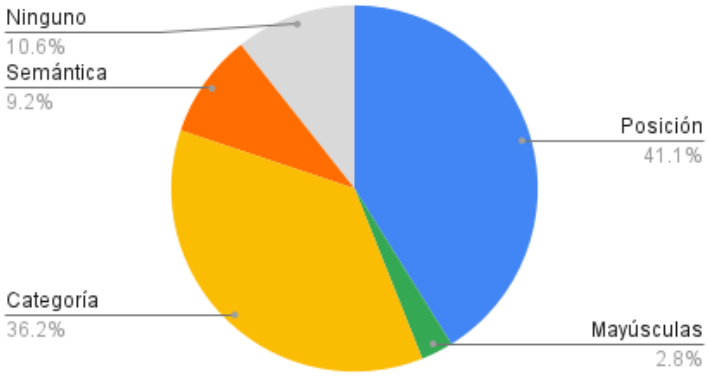

Fig. 4. Porcentaje de participantes que manifestaron preferencia por cada uno de los cuatro indicadores en el primer análisis.

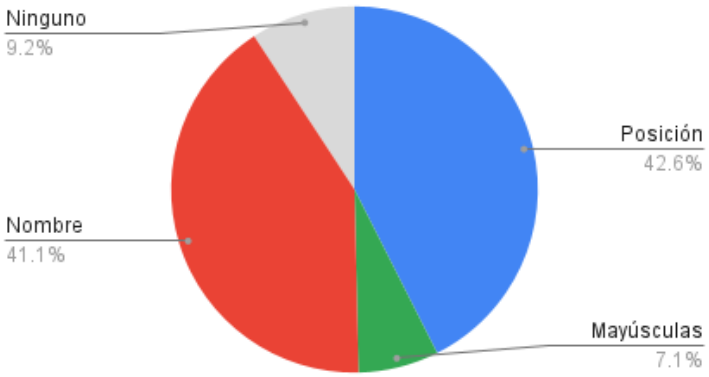

Fig. 5. Porcentaje de participantes que manifestaron preferencia por cada uno de los tres indicadores en el segundo análisis.

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. En el caso de la preferencia definitiva, observamos que 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.

Esto nos muestra 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. 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. 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 6.

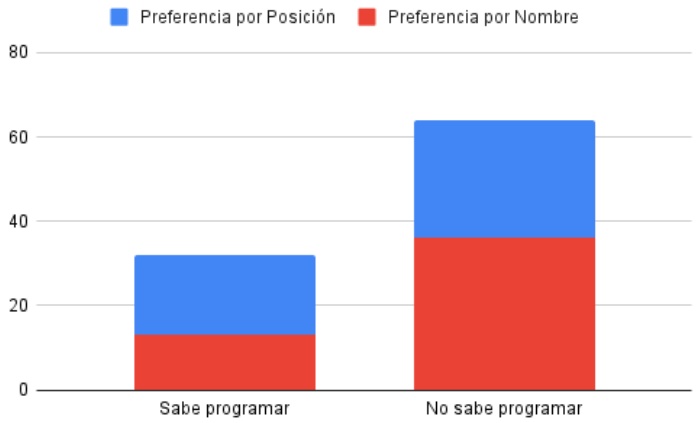

Fig. 6. Cantidad de personas que manifestaron preferencia por alguno de los indicadores *Posición* o *Nombre*, agrupadas según si afirmaron o negaron saber programar.

Observamos que no parece haber correlación entre autopercibir 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 el gráfico anterior, 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 todes les 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.

5 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 todes les participantes tuvieron que cursar la materia correlativa EPyL, observamos que los contenidos impartidos no alcanzan para que les 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 esté 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.

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.

References

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 10○ 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, Mattthew 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).