Int. J. Human–Computer Studies 148 (2021) 102577

Contents lists available at ScienceDirect

International Journal of Human - Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

Exploring the first experiences of computer programming of older people with low levels of formal education: A participant observational case study

Sergio Sayago [a,*], Ángel Bergantiños [b]

[a] Universitat de Lleida, Spain
[b] Universitat de Barcelona, Spain

ARTICLE INFO

Keywords:
Computer programming
Older people
Participant observation
Textual and visual programming languages
Social inclusion

ABSTRACT

Computer programming is widely regarded as a key skill in the 21st century. Yet, and despite a growing aging population and interest in promoting computer programming for all, research on this topic with older people (60+) is scant in the Human-Computer Interaction literature. This paper presents a qualitative case study aimed to explore the first experiences of computer programming of a group of older active computer users with low levels of educational attainment (i.e., primary school / K-12). Over a 6-month period, we provided a hands-on introduction to several textual and visual programming languages and environments to ($N = 29$) older and adult people in three courses in an adult educational center. We reveal and explain relevant factors that shape, and help us understand, the participants' computer programming learning experiences, including their motivations, difficulties, and identity, along with strategies that hindered and fostered empowerment. Implications for research and design are discussed.

1. Introduction

In recent years, there has been a surge of public interest in promoting computer programming for all. Examples are free massive online programming courses (e.g. Coursera, edX), along with initiatives such as *Code Week*[1] and *All You Need is Code*[2] in Europe, and the *Hour of Code*[3] in the U.S. This public interest in fostering programming for all is due in part to the fact that programming, i.e., the process of writing computer programs, is widely regarded as a key skill in the 21st century (Rushkoff, 2010; Kafai and Burke, 2014; Vee, 2017), and programmers as the "lifeblood of our technical society" (Halvorson, 2020, p. 8). Another significant push for programming for all comes from the maker movement (Kafai and Burke, 2014).

This paper addresses computer programming with older people (60+). Why is this issue important? Older adults represent a large and fast-growing fraction of the global population. By programming, they can create or modify software and tangible applications that are related to their interests and needs (Guo, 2017). This can result in more accessible and useful technologies for a growing aging population, as most of today's technologies have been designed without considering older people (Newell, 2011). Programming can also be a gateway for older people, especially those who do not have the knowledge or resources needed (e.g., low literacy levels, living in working-class neighborhoods that lack relevant resources), to participate in the maker movement, which has been criticized for its lack of demographic diversity (Meissner et al., 2017; Baudisch and Mueller, 2017; Tanenbaum et al., 2013). Being able to read and comprehend code also provides older people with an opportunity to better understand, question, and participate in today's society, as programming is shaping it (Kitchen and Dogde, 2011) and looking more and more like a new literacy (Vee, 2017).

This paper presents a case study aimed to explore the first experiences of computer programming of a group of older active computer users with low levels of formal education (i.e., primary school / K-12). We focus on this profile of older people because our long-term research goal is to empower (i.e., to extend their abilities and develop new skills (Schneider et al., 2018)) those older adults who are running, or run the risk of, lagging behind from computing revolutions. Older people in higher socioeconomic groups and well-educated use digital technologies at higher rates than those in lower groups (PEW, 2014; Schehl et al.,

* Corresponding author.
E-mail address: sergio.sayago@udl.cat (S. Sayago).

[1] Code Week. Retrieved October 13, 2020 from http://codeweek.eu/
[2] All you need is code. Retrieved October 13, 2020 from http://www.allyouneediscode.eu/
[3] https://code.org/ Retrieved October 13, 2020.

https://doi.org/10.1016/j.ijhcs.2020.102577
Received 14 July 2020; Received in revised form 19 October 2020; Accepted 8 December 2020
Available online 4 January 2021

2019). Whilst research on computer programming is vast, previous works addressing this topic with older people are scant – see Section 2. Against this background, we decided to carry out an exploratory case study, based on participant observation (DeWalt and DeWalt, 2011), intended to (a) discover and examine in-context the first encounters with programming of older people with low levels of formal education, by providing them with a hands-on introduction to different programming languages and environments, and (b) draw from their experiences a number of relevant factors that shape, and account for, their relationship with programming, and that can inform future studies.

The study took the form of three courses conducted over a 6-month period in an adult educational center in a working-class neighborhood of Barcelona (Spain), wherein older and adult people participate. All the courses in this center are open to all its participants. Consequently, although our study focused on older people, we opened it to adult people, and doing so helped us enrich the study, as we discuss later. In our study participated older (70+: 5; 61–70: 11), middle-age[4] (40–60: 7), and young (18–39: 6) non-English adult speakers with several cultural backgrounds (Spain, Eastern Europe, Latin America, Asia and Arabia), low levels of educational attainment (approx. 80% finished primary school, and 20% secondary school), and previous experience of using computers. None had previous experience of programming. We provided them with a practical introduction to textual and visual programming languages and environments, i.e. Java, Python, Processing, Scratch, and App Inventor. The choice of the languages was motivated by the goals of the study and their potential usefulness for the participants. We discuss this issue further in Section 3.

While this paper reports a single case study, which was conducted with a specific profile of participants, we believe its findings have generalizable research value to the field of Human-Computer Interaction (HCI). We discuss their motivations for exploring computer programming, which include feeling more socially included and competent, and learning more about how computers work. We also show that their most important difficulties in learning programming were mostly cognitive-related, and indicate that these difficulties were remarkably similar among the participants, regardless of their age or cultural backgrounds. We also show that the participants finished the courses by being able to read, understand, and write simple programs, challenging stereotyped (mostly negative) views of older people and digital technologies. This paper makes the following contributions:

- An exploratory case study of the first experiences of computer programming of a group of older and adult active computers users with low levels of formal education and several cultural backgrounds
- Relevant factors that shape, and help us understand, the computer programming learning experiences of a group of older and adult active computer users with low levels of formal education, such as their motivations, difficulties, and identity
- Implications for research and design, especially related to understanding better older people as technology users, co-creating useful, user and learner-centered instructional materials, and designing better (more inclusive) tools for programming.

2. Related work

In 2.1 we present a progression in HCI research with older people, from consumers to producers of digital content, which is relevant for this study. In 2.2 we review previous works on computer programming with older people.

2.1. From consumers to producers of digital content

aging has recently become a significant research area in HCI (Vines et al., 2015; Sayago, 2019). Much of this research regards older people as consumers of digital content (Guo, 2017). Yet, there is a progression in research that positions them as producers of digital content and artefacts. We are witnessing older people who are blogging (Brewer and Piper, 2016), making custom electronics (Jelen et al., 2019), wearing smartwatches and generating quantified self-data (Rosales et al., 2017), creating digital games (Sayago et al., 2016), producing digital videos (Ferreira et al., 2017), and engaged in crowd work (Brewer et al., 2016). This transition from consuming to generating digital content and artefacts is of direct relevance to our work, as older people can program entirely new software applications on their own by writing code (Guo, 2017). This transition can also contribute to debunk widespread (and mostly negative) stereotypes of older people and their interaction with digital technologies (Durick et al., 2013). This paper aims to extend the progression that positions older people as producers of digital content by adding further and new evidence to it, by addressing computer programming.

2.2. Computer programming with older people

Computer programming attracts a lot of research. Previous works have developed programming languages and environments that lower the barriers to programming (Kelleher and Pausch, 2005; Sim and Lau, 2019). Other studies have examined error and notification messages (Becker et al., 2019), debugging strategies (Murphy et al., 2008), emotions (Kinnunen and Simon, 2010), practices of programming (Bergström and Blackwell, 2016), and programming patterns in block-based and text-based programming languages (Weintrop and Holbert, 2017). Previous research has also put forward pedagogical strategies for teaching children to code (Bers, 2019), examined the difficulties non-English speaking people experience while learning programming (Guo, 2018; Pal, 2016; Dasgupta and Mako, 2017; Vogel, 2020), analyzed the benefits of learning a visual programming language over a traditional text-based language (Noone and Mooney, 2018; Weintrop and Wilensky, 2017), reported on what programming languages developers use and why (Pang et al., 2018), and examined creative coding (Li et al., 2020). Despite a growing aging population, very little of this research has been conducted with, or has considered, older people.

From the title and abstract of the papers published in the three most recent proceedings (Malizia et al., 2019; Barbosa et al., 2017; Díaz et al., 2015) of *End-User Development*[5] and the *IEEE Symposium on Visual Languages and Human Centric Computing* (VL/HCC, 2018–2020),[6] none of them addressed older people. We also conducted keyword searches (see Table 1) in the ACM Digital Library (DL), SCOPUS, and IEEE Xplore. We selected these three databases because they are particularly relevant for this paper, covering a broad range of studies in the fields of Aging, HCI, and Computer Science. To keep our search as broad as possible, we applied our search string to the fields *title, abstract,* and *keywords* in SCOPUS, *anywhere* in the ACM DL, and *all metadata* in IEEE Xplore. We searched for papers written in English and published in peer-reviewed scientific journals and conference proceedings within the last ten years (in the period from 2010 to 2020). We excluded papers that did not explicitly deal with (a) computer programming, and (b) older people (60+). A total of 360 papers were found in the ACM DL, 293 in SCOPUS,

[4] According to Encyclopedia Britannica (https://www.britannica.com/science/middle-age), middle age is generally defined as being between the ages of 40 and 60.

[5] whose goal is to empower end-users who are not necessarily experts in software development to create or modify their software to address their own specific needs

[6] whose mission is to support the design, theory, application, and evaluation of computing technologies and languages for programming, modeling, and communicating, which are easier to learn, use, and understand by people

Table 1
Keyword search in ACM DL, SCOPUS, and IEEE Xplore.

In ACM DL:
- 290 Results for: [All: "senior"] AND [All: "computer programming"] AND [Publication Date: (01/01/2010 TO 12/31/2020)]
- 35 Results for: [All: "elderly"] AND [All: "computer programming"] AND [Publication Date: (01/01/2010 TO 12/31/2020)]
- 11 Results for: [All: "older people"] AND [All: "computer programming"] AND [Publication Date: (01/01/2010 TO 12/31/2020)]
- 24 Results for: [All: "older adults"] AND [All: "computer programming"] AND [Publication Date: (01/01/2010 TO 12/31/2020)]

In SCOPUS:
- 115 documents for (TITLE-ABS-KEY ("senior") AND TITLE-ABS-KEY ("computer programming")) AND PUBYEAR > 2009
- 114 documents for (TITLE-ABS-KEY ("elderly") AND TITLE-ABS-KEY ("computer programming")) AND PUBYEAR > 2009
- 21 documents for (TITLE-ABS-KEY ("older people") AND TITLE-ABS-KEY ("computer programming")) AND PUBYEAR > 2009
- 43 documents for (TITLE-ABS-KEY ("older adults") AND TITLE-ABS-KEY ("computer programming")) AND PUBYEAR > 2009

In IEEE Xplore:
- 6 results for (("All Metadata":"senior") AND "All Metadata":"computer programming"), 2010–2020
- 1 result for (("All Metadata":"elderly") AND "All Metadata":"computer programming"), 2010–2020
- No results found for (("All Metadata":"older people") AND "All Metadata":"computer programming"), 2010–2020
- 1 result for (("All Metadata":"older adults") AND "All Metadata":"computer programming"), 2010–2020

and 8 in IEEE Xplore. Two papers met the eligibility criteria.

(Guo, 2017) is "the first known study of older adults learning computer programming" (p. 7070). It is based on an online survey on the motivations, learning practices, and frustrations of approximately 500 English-speaking older people (over 90% were Managers, Professionals or Technicians), from 52 different countries, who were learning programming by using an educational website. The results show that making up for missed learning opportunities during youth, keeping their brains challenged, and implementing a specific hobby project idea were the respondents' top three motivations for learning programming. The respondents also reported using free online resources, mostly MOOCs, blogs, and web tutorials. The three most important reported learning frustrations were bad pedagogy, cognitive impairments, and no human contact with tutors or peers.

(Ohashi et al., 2020) reports the curriculum of a programming course case study of senior citizens in Japan. The ultimate goal was to train older people as lecturers of computer programming in elementary schools. Two courses were conducted (5, 2-h long sessions a week) with 30 older people. Scratch was chosen as a programming language, and teaching materials used for elementary school pupils were revised and reused. Based on questionnaires, most of the participants' motivation was to be able to use computer or programming, and to communicate with children or grandchildren. In terms of considerations to instruction design for senior citizens, scheduling and pacing, and diverse learning styles were found to be key.

This paper extends these two studies as follows. Firstly, while the sample of older adults who participated in (Guo, 2017) was skewed towards highly educated, technology-literate, self-motivated, and English-speaking people, the older adults who participated in this study are non-English speaking people with low levels of education and different cultural backgrounds. The profile of participants in (Ohashi et al., 2020) consisted of older Japanese people. Secondly, this paper addresses several visual and textual programming languages and environments, compiler error and notification messages, differences and similarities in programming learning between younger and older people, and instructional approaches, in addition to motivation and learning frustrations, by combining first-hand observations with conversations over a 6-month period.

3. The case study

In order to achieve one of the objectives of the study, i.e., to discover and examine in-context first experiences of programming, we deemed it important to capture how these experiences are created and experienced by the study's participants. To this end, we conducted participant observation (DeWalt and DeWalt, 2011), which is a method to collect data in naturalistic settings by carrying out first-hand observations of, and taking part in, the activities of the people being studied. We conducted the study in Àgora (AG), an adult educational center in Barcelona, Spain.

AG has been operating for almost 40 years. Since the 1980s, AG has been fostering the social and digital inclusion of people who are, or might be, excluded from the Catalan society, such as immigrants and older people. To do so, AG adopts an intergenerational dialogical learning approach (Sánchez Aroca, 1999), which empowers the students – using AG terminology, *participants* – to decide what they want to learn in free courses. Their decision is usually based on the needs they aspire to fulfill in their everyday lives. Participants regard digital technologies as instrumental in fostering inclusion. Courses on computing and the Internet take place daily and are mostly attended by older participants. Volunteers, who are mostly older people that became fairly independent computer users by enrolling in courses in AG, help to run the courses. Other volunteers are Bachelors, Masters, PhD students, and postdocs conducting academic fieldwork activities.

The case study took the form of in-person courses on programming (Fig. 1). The courses (including the curriculum and materials) were designed and conducted by the authors of this paper. As fieldworkers are key research instruments in participant observation (and ethnography) (Coffey, 1999), and part of the social world they study (Hammersley and Atkinson, 2007), we disclose key aspects of our identity related to this study in an attempt to clarify who the *we* this paper mentions is and how that influences the research (Schlesinger et al., 2017). Both of us have a background in Computer Science (CS). One of us (the first author) holds a PhD in CS and HCI, has previous experience of teaching computers and the Internet to older people, and programming to first and second-year students of CS. The second author was an undergraduate student (of CS and Mathematics) at the time of doing this study.

We provided the participants with a hands-on introduction to Java, Python, Processing, Scratch, and App Inventor. As stated in (Weintrop and Wilensky, 2015b), a longstanding question faced by computer science educators is what language to use to introduce learners to programming. In our study we took a number of factors into consideration (Gupta, 2004) – in addition to, perhaps inevitably, our own personal experience of programming - to choose the aforementioned programming languages and environments. The factors were the exploratory goal of the study, the profile of the participants, and the potential usefulness of the programming languages and environments for them.

Although Java, Python, and Processing programs can be quite complex to understand and create for novices, especially for those who do not either speak or read English, we chose Java and Python because

Fig. 1. Participants in a course.

both are very popular programming languages (TIOBE, 2020) and highly demanded in the job market.[7] This could help our participants to (i) access a lot of online resources and communities, and (ii) provide them with employment opportunities, especially the younger ones. Java is connected (via Android) with mobile apps and smartphones, which are popular devices amongst older people who are online (Rosales et al., 2017). Yet, we did not focus on Object Oriented Programming in our study, as we discuss in 3.1 and 3.2. Python allows us to teach programming concepts in an easy to understand manner to young students (Noone and Mooney, 2018). Based on our own experience of programming in Python, this programming language could also be a viable choice when it comes to introducing older and adult people with low levels of formal education to computer programming, as it might have low barriers to startup (e.g., easier syntax than Java). With respect to Processing, we used it in our study because of its connection with creative coding, which could appeal to our participants, due to its visual and artistic aspect, and motivate them to exploit their creativity. Processing is a language for learning how to code within the context of the visual arts and is being used by artists, designers, researchers, and hobbyists for learning and prototyping.[8]

While Java, Python, and Processing are text-based programming languages, we explored block-based programming languages and environments, namely Scratch and App Inventor, because block-based programming languages "are increasingly becoming the way that novices are being introduced to the practice of programming and the field of computer science more broadly" (Weintrop and Holbert, 2017, p. 633). Programming in these environments takes the form of dragging blocks into a composition area and snapping them together to form scripts, thus helping to alleviate difficulties with syntax. Scratch is a very popular block-based programming language, with an active online community around it (Resnick et al., 2009; Kafai and Burke, 2014), and can foster intergenerational activities (e.g., older people and grandchildren). App Inventor is based on Scratch and connected to mobile app development (Wolber et al., 2014; Ortega et al., 2017). Scratch and Python are also connected to the maker movement.[9]

In 3.1 we present the approach followed in the courses for introducing the participants to programming. Afterwards, we provide an overview of the courses. In 3.3 we present the profile of the participants, and recruitment method. Lastly, we present the data gathering and analysis approach adopted in the study, and ethical aspects.

3.1. The approach

In addition to the programming language, previous works have highlighted that how programming is introduced and taught matters (Cunningham, 2018; Mohorovičić et al., 2011). In our courses, it was important that a user-centered and tested framework for technology learning of older adults (Sayago et al., 2013), and the principles of the learner-centered design process (Guzdial, 2016) were utilized. By drawing on a four-year ethnographic study of ICT learning (programming was not included) with 420 older people in an adult educational center in Barcelona and a computer clubhouse in Dundee (Scotland), (Sayago et al., 2013) makes explicit two key and distinguishing aspects of older adult ICT learning: (i) life-centered (e.g., strong connection with their lives – what they consider they want to need to know) and (ii) life experience (e.g., lessons learned over a person's lifetime). Both aspects make their learning experience different from that of children (Knowles et al., 2005). (Guzdial, 2016) is a learner-centered design approach intended to create computing education for a broad audience. Central to this approach is that programming is not just for the professional software developer, and respect for the learner, i.e., we need "to construct learning opportunities for who the *learner* is and wants to be, not for the *expert* that we computer scientists want them to be" (p. 16). The principles of learner-centered design include understanding where the learners are starting from, what they want to do, and where they are likely to have trouble, as well as expecting the learners to change and using a language they understand.

We found it difficult to anticipate key aspects of the principles proposed by (Sayago et al., 2013) and (Guzdial, 2016), especially in the first course, such as the relationship between programming and our participants' life experiences, interests and needs, the language we had to use, and the type of authentic programming learning activities we should design. We did not have previous experience of teaching programming to this user group. We did not find empirical evidence from our review of previous and related works either. We did find approaches, adopted with other groups of novice programmers (e.g., high school students), that zero in on very specific issues, such as metacognition (Loksa et al., 2016). Albeit important, we decided to adopt a more open-ended approach to achieve the objectives of the study. Thus, we decided to fall back on our experience of teaching programming in the traditional STEM perspective. Although this way of teaching programming has drawbacks, mostly due to its syntactic approach (Cunningham, 2018) and little attention to cognitive aspects of programming (Loksa et al., 2016), it is fairly general and consolidated (Bers, 2019). As opposed to (Ohashi et al., 2020), we did not consider adapting materials used for elementary school pupils, as doing so could have been paternalistic and strengthened already negative views of older people (Riddell and Watson, 2014). Instead, we followed tips for teaching programming "at any level and to any audience" (Brown and Wilson, 2018, p.1), such as live coding, making predictions (i.e., ask students what they think a program would do), and pair programming, along with instructional materials developed by ourselves, and online tutorials and guides freely available online (e.g., tutorialspoint.com).

After explaining the basic structure of a program and having the participants write (or create) the typical *Hello World* program (in the different programming languages of the courses), we addressed important elements of procedural programming, such as variables, operators, iterative, and conditional statements. In the case of Java, we did so within the main method of a program consisting of one class. We ran practical sessions, as programming skills are acquired and improved with practice. In the sessions, we conducted live coding, to show the participants how a program can be created step-by-step, and asked the participants to guess the output of a given program, to improve their programming reading skills. The sessions were in Spanish, as all our participants read, understand, and speak in this language. Some participants, especially those adults from Eastern Europe, Latin America, Asia and Arabia, had difficulties communicating in Catalan. We translated keywords that programming languages such as Java, Python, and Processing use in English into Spanish in the sessions (e.g., print / imprimir, for / para, if / si) so that participants got used to, and learned, these new words for them. The participants did a number of traditional programming exercises in STEM (e.g., writing a program to check if a number is odd or even) by programming in pairs and solving Parsons Problems, where chunks of code have to be placed in the correct order, to help them learn further about the program flow. We also provided participants with programs that had gaps (e.g., a function to complete) for them to fill in, so that they could concentrate on specific aspects, avoiding more complex ones (e.g., input / output), and, at the same time, look at and become more familiar with the code of a program.

3.2. The courses

We conducted three courses (A, B, and C). Table 1 provides an overview of the activities conducted in them. Each course had the same duration (3 months) and format (weekly sessions of 2-h long) as the

[7] https://www.computer.org/publications/tech-news/trends/programming-languages-you-should-learn-in-2020 Retrieved October 13, 2020

[8] https://processing.org/ Retrieved October 13, 2020

[9] http://www.makerspaceforeducation.com/coding.html Retrieved October 13, 2020

other courses on computers in AG. Course B and C were conducted during the same months, in morning and afternoon sessions, respectively. Conversations with members of the board of AG revealed that these were the first courses on computer programming in the center.

Course A was devoted to Java. Course A was the first course we conducted in the study and we decided to focus on a single and textual programming language. We also considered Python. Yet, since Java and Python are both in English, our participants are not English speakers and had no previous experience of programming, and we had more experience of programming with Java at the moment of running this course, we focused on it. Participants wrote a number of basic and traditional programs in the STEM context, such as counting the number of vowels in a word, using variables, loops (e.g. *for* and *while*), conditionals, and input / output, in a single main program. Participants also created an interactive program with a Graphical User Interface (GUI) in Swing with the multiplication tables. Participants considered that a visual program such as this could be useful for them (to brush up on their mathematical knowledge and play games with their grandchildren). Rather than writing the program from scratch, we provided them with a pre-written program with gaps they had to fill in. The program had the code in Swing of the GUI and the code to complete was related to the generation of the multiplication tables, thus avoiding OOP skills. Participants could change the look-and-feel of the GUI if they wished through Netbeans, the IDE (Integrated Development Environment) used in the course. At the end of the course, we introduced participants to Scratch to get their opinion on a different, block-based, programming language, an help us inform the design of the following courses.

Course A informed the design of Course B and C. Course A threw (some) light on the relationship between the key aspects of (Sayago et al., 2013) and (Guzdial, 2016), and programming with our participants. As we discuss further, and contextualize, in the results section, we identified programming exercises that participants were willing to write, such as calculating the number of hours a person has lived, and those that had a connection with their lives, i.e., programs that they could show to and use with others at home or in the center, such as online games and mobile apps. Traditional STEM exercises were not very motivating for them, perhaps, due to the mathematical aspect and little connection with their interests. We also realized that devoting three months to a single programming language was difficult to keep participants motivated, as they struggled to write programs independently and made numerous errors, leading to feelings of incompetence and social exclusion. Thus, we considered that diversity could be an alternative to have them more engaged. Course A also confirmed that live coding, providing the participants with programs with gaps they had to fill in, and pair programming, were useful and engaging ways to introduce them to programming and explore their programming learning experiences.

Taking the lessons learned from Course A into account, Course B and C provided participants with a hands-on and basic introduction to Python, Scratch, App Inventor, and Processing. Three sessions were devoted to each programming language (see Table 2). The number of the programming languages might be judged as too much or distracting. Also, three sessions is not enough to learn a programming language. Yet, given that the aim of this study was exploratory, and taking into account the lessons learned from Course A, we considered that it was worthwhile to have the participants read and write basic programs in different programming languages, and that three sessions could allow us to do so. As Table 2 depicts, participants wrote, in some cases, programs with a certain level of complexity. Figs. 2–4 show a sample of computer programs written during the courses by the participants.

While Java, Python, and Processing are in English, Scratch and App Inventor can be used in other languages than English. We set Scratch and App Inventor to Spanish. We did not use them in English because the participants' level of English was not good enough to use these tools in that language. We considered using Scratch and App Inventor in Catalan. However, some of the participants had problems communicating in Catalan. Also, App Inventor was not available in Catalan at the time of doing the study. Hence, as we wanted to use both tools and run inclusive sessions, we decided to use Scratch and App Inventor in Spanish. Still, some participants, those who speak both Spanish and Catalan, did use Scratch in Catalan, as a personal preference.

Table 2
Overview of the courses: programming languages, environments, and activities.

Programming languages / environments	Overview of the activities
Java	The first four weeks were devoted to an introduction to the NetBeans IDE (Integrated Development Environment), key programming aspects, such as conditional and iterative statements, and to do practical exercises in the STEM tradition. The remainder of the sessions (8) was devoted to the program of multiplication tables, in which participants had to write the code (i.e., iterative statements – for / while - and multiplications – number * index) to calculate and generate the tables.
Python	The first session was devoted to Jupyter Notebook, how to install it at home, and the basics of Python. The other two sessions were devoted to writing programs related to aspects in which the participants were interested, such as calculating the age or figuring out the zodiac sign of a person given his or her date of birth (without considering issues such as leap years).
Scratch	Pong game. We provided the participants with the structure of the game and asked them to write the code that was missing in three parts: scoring, movement of the ball, and movement of the pales. The first session was devoted to creating an account in Scratch, introducing the online programming environment, and the Pong game. The other two sessions were devoted to writing the missing parts of the code provided. All participants finished the game.
App Inventor	The three sessions of App Inventor were devoted to the development of a simple app that allowed participants to convert Euros to another currency. Participants brought their smartphones, installed a QR reader into them during the sessions, and interacted with the app in their own phones. All participants finished the app.
Processing	In the first session, we provided the participants with the structure of a program that allowed them to draw basic geometric figures (e.g. circle, square) in a canvas. We also introduced them to the Processing language and IDE. In the remainder of the sessions, we encouraged the participants to 'show us their creativity', by completing the code to modify the figures and create new, more complex ones. Participants modified the program to draw, for example, the skeleton of a person and a chair.

3.3. The participants: Recruitment and profile

We recruited the participants by following the procedure of AG. We prepared a short description (in the local language) of the courses, which were announced on the school's bulletin board, along with the other courses in the center. As other public centers in the local area provided official programming courses (with a certificate of completion), we deemed it important to highlight the explorative, research-oriented, free of charge, and hands-on introduction to computer programming features in ours. The (translated) text of one of the courses was:

"*Are you an active computer user interested in writing your own programs? Have you heard about programming and do not have a clue about what it means? There is a new course in the school, "programming 3.0". A researcher and lecturer in CS at a public university will run it. The course will provide you with a hands-on introduction to popular programming languages. You will also contribute to research with your opinions and experiences. Are*

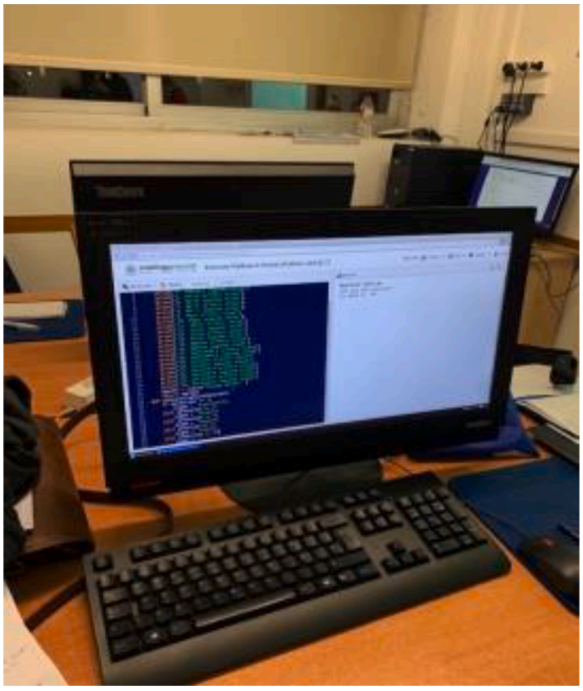

Fig. 2. Program in Python that calculates the zodiac sign of a person given her year of birth.

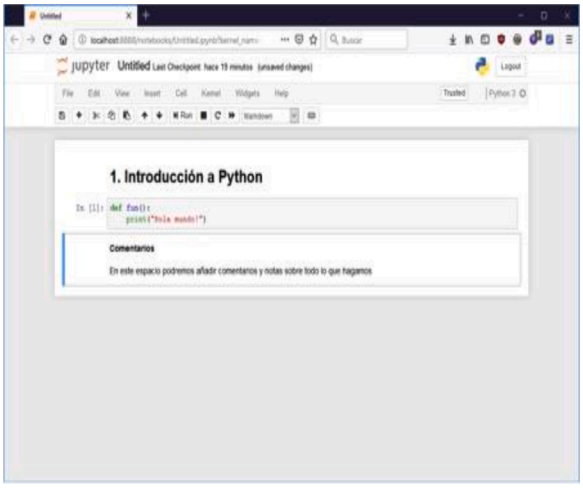

Fig. 3. Jupyter Notebook with a 'Hello World' program.

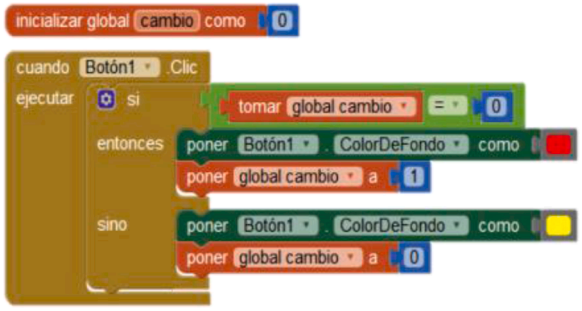

Fig. 4. Code (in Spanish) in App Inventor of an app with a button.

you going to miss it? Sign up in the secretariat!"

From the announced list of courses with short description, participants interested in the course/s sign up for them in the secretariat, wherein they are informed about the course and the person who organizes it. The inclusion / exclusion criteria of our courses were the same criteria as those followed in the other courses on computing in AG: if the participant considers that he or she has the level of experience with computers to follow the course, and there are places available, they take part in it. Their participation is always voluntary.

Our conversations with the participants ($N = 29$; Men: 18. Women: 11) indicated that they were active computer users, non-English speaking people, had low levels of educational attainment (approx. 80% finished primary school, and 20% secondary school), and were original from Spain (24), Eastern Europe (2), Latin America (1), Asia (1) and Arabia (1). All of them owned smartphones and were active WhatsApp users. None of them reported having previous experience of programming. Table 2 shows the distribution of participants in the courses. No participant took more than one course. Two-thirds of the participants (19) attended all sessions (12 for each course). The participants spent in each class between 1.5 h (due to other commitments, such as taking care of their children, grandchildren, or having an appointment with their GP) and 2 h (Table 3).

3.4. Data gathering and analysis

We jotted notes of our observations and conversations immediately after the sessions (36), which were so active that hindered in situ note-taking. The first author did the note-taking in course A, while the second author did so in courses B and C, as we could not participate together in all the sessions. The first author participated in some sessions of courses B and C. To strengthen this aspect of data gathering (i.e., avoid possible biases in writing fieldnotes post hoc), we met regularly (once a week) to share experiences and lessons learned from the sessions, discuss progress and different perspectives, and deal with unexpected issues (e.g., difficulties installing software or Internet connectivity). In our office, we wrote descriptive notes in a shared, online (text) document. We did not use video cameras in any course, as doing so could have been highly disruptive. However, we did take pictures (Figs. 1–4 of this paper) by using our smartphones, which is a technology our participants already brought to, and used, in the courses.

We analyzed the notes by conducting thematic analysis (TA), in particular, reflexive thematic analysis. Given that TA is as an umbrella term (Braun and Clarke, 2019), with three different schools (coding reliability, codebook, and reflexive TA), we deem it important to clarify which one we adopted, and why, as each school has its strengths and limitations. We choose reflexive TA because, in our opinion, it fits well with participant observation (DeWalt and DeWalt, 2011) and case studies (Lazar et al., 2017), as it emphasizes meaning as contextual or situated, and researcher subjectivity as a resource (Braun and Clarke, 2019). In familiarization (phase 1), we read the notes, at the end of the courses, to look for possibilities, connections and potential interesting ideas. In phase 2, we generated codes by conducting a systematic identification of meaning through the notes. We constructed the main themes in phase 3 by collating codes into potential themes that were related to the objective of the study. We wrote memos to revise and

Table 3
Profile of participants in the courses.

Course	# participants	Age range	Men/ Women
Course A	6	70+: 3; 60–70: 3	4 men 2 women
Course B	12	70+: 1; 60–70: 4; 50–60: 5; 40–50: 0; 30–40: 0; <30: 2	6 men 6 women
Course C	11	70+: 1; 60–70: 4; 50–60: 0; 40–50: 2; 30–40: 2; <30: 2	8 men 3 women

re-define themes (phase 4), and shared the results with colleagues to gather their feedback and external perspective (phase 5). We wrote the manuscript in phase 6. We conducted between 3 and 5 iterations in phases 3 to 6 until the themes could tell a compelling interpretation of the data.

Each of us conducted the analytic process outlined above independently to get two perspectives of the programming experiences we had witnessed in the courses. The notes of Course A were analyzed by the first author, as the second one had not participated in it in any way, while the notes of the other courses were analyzed by both of us. In Phase 5, we put together and discussed the results, and shared the memos among some of our colleagues, who had not participated in the courses. In keeping with reflexive TA, we did not aim for 'consensus coding' (Braun and Clarke, 2019). Instead, we aimed for a credible and rigorous interpretation of our participants' experiences of programming. The analysis finished when the results achieved that goal, in the authors' opinion.

3.5. Ethical aspects

Ethical approval was granted by the Council Center, the decision-making body of AG. Participants granted us written consent to use extracts of our notes in publications related to the study, and pictures of themselves in the courses, and use this material freely and without any restriction in publications related to the courses.

4. Findings

The analysis outlined in Section 3 yielded five main themes, which we use to organize and present the results. We use extracts taken from the fieldnotes to include the participants' voices in the results. These extracts have been translated into English by the authors. Participants are identified by codes, e.g. [P12, AGE]. We include brief analytic commentary sections to reflect on and discuss key aspects of the results. To simplify, the voices of the authors are presented in the first person of the plural (i.e., we).

4.1. Motivations for programming: learning about computers, feeling more socially included and competent, and creating something useful

Our participants' main motivations for exploring computer programming were to: know more about how computers work, e.g. "*Great! I always wanted to **learn** how these things work and are made*" [P1, 65] feel more socially included and competent, as the following conversation between two participants we overheard before a session was due to begin show

"*When you go to a computer shop, and they see you (and older person) coming… you know, you feel like **fish out of water**. When you grow old, you learn that people treat you differently, because of your age"* [P12, 68] *(…)*

The same when you're an immigrant. It's sad, but this is the real world… This is something you learn over your lifetime, this is NOT only about age" [P4, 40] *(…)*

*By learning programming, we could speak a language techy people know about and make better, more informed decisions. To speak with property – I mean, showing that you know what you are talking about - and **not like a fool***" [P12, 68] create useful or interesting applications for them and others, e.g. "*the program you showed us turns someone's age into days! It's really amazing. Will we write it? I could **create** something nice for my son, he is learning basic math now*"[P4, 40]

4.1.1. Commentary

Computational literacy, one of the least popular motives for learning to program of the respondents in (Guo, 2017), was our participants' main motivation for learning computer programming, along with social inclusion. This difference might be due in part to their very different profiles.

Using computer programming as a vehicle for improving (perceived) social inclusion reinforces the idea of 'coding is connecting' in programming with children (Kafai and Burke, 2014). As opposed to them, for our participants connecting does not mean to 'be seen'; it means to 'feel more socially included'.

It might be surprising that finding a job was not listed as one of the motivations, especially for the younger participants. In light of the results presented below, this might have been due to the difficulties they experienced in their initial encounters with computer programming.

4.2. Lots of difficulties, mostly cognitive-related, with important meanings behind, related to the identity of the participants: struggle for independence, social inclusion, and feelings of incompetence

"*I'm sweating. Geez! This is really annoying. It says there is an error, on that line, but I don't see it. Where is the damn error? This programming thing is very picky. If you fail to write a parenthesis, or miss a semicolon somewhere, the program falls apart. It's quite sad to realize that you are able to use your smartphone, edit pictures and watch movies online **on your own**, and then see that you're completely useless at programming*". [P1, 65]

Participants paid a lot of attention to what we said and the programs we wrote on the whiteboard. We explained to them the meaning of keywords that programming languages use in English (e.g., for / para, while / mientras) in Spanish. They wrote extensive paper-based notes, as "*otherwise I won't remember. Where did you say the icon to run the program is? What does this word mean? Oh, I see… I need to write that down too. **My notes are my memory***" [P6, 70]. We told them that they could read the documents with the instructions we had prepared for the courses (mostly PowerPoint presentations), and that we used in the sessions. Participants thanked us for preparing this material, which they checked occasionally in the sessions. However, participants relied on interacting with us and on their notes throughout the study. We also encouraged them to check tutorials, code examples, and how-to guides online (e.g., (https://www.tutorialspoint.com/index.htm). They did try, but "*this is like reading Chinese to me. This is very advanced. We need something much clearer, more guided and understandable. You see, my notes, I do understand them. We can't learn programming by reading tutorials. We need a teacher.*" [P13, 55]

Participants made errors in all the programming languages and sessions. Our close interaction with them enabled us to realize that the errors were mostly: syntactically based. This was especially significant with Java, "*If you miss the semi colon… kaput!*" [P21, 50] Participants made fewer errors with Python and Processing, "*The colors help a lot. But if you miss a parenthesis, or the next line is not indented…the program falls apart! It's very picky. This might **be too much for us**. **We will never be like others**, who can do these things*" [P3, 66] due to difficulties using the mouse, especially older participants, struggled to move blocks in Scratch and App Inventor, (see Fig. 4), "*You don't write, fair enough, but you've to move this tiny block across the screen and put it in the right place, which isn't easy with old hands.*" [P1, 65]

The messages / notifications of the compiler were of little use for them. We observed that participants did not know how to solve the errors by reading the compiler error messages, which they read carefully

This says that 'computer' does not exist. This can't be true. I did write the variable with the name 'computer' [P2, 40]

(Instructor) *This message is telling you that the word 'computer' has been misspelled. See, you missed the r here.*

I see! The computer could have told me so!" [P2, 40]

The language of the message was not an issue. We observed that most of them used online tools (i.e. Google Translate) to translate the messages – and websites - in English. The younger participants had already developed this practice, while the older ones learned it from them in the courses. Participants paid scant attention to visual marks (such as a dotted or curved line in red) when there was an error, because of practices they had developed with other tools:

(Instructor) *You see, this text with this line here means that there is*

something wrong.

Oh, I see, I thought ***it was like Word****, when you write a word that the computer does not understand (like my first name) but that it is not wrong. I ignore it.*" [P5, 60]

We observed that younger participants were 'faster learners' than the older ones. They were able to correct syntax-based errors (e.g., a missing semicolon) quicker. They also seemed to follow our instructions faster, as we did not have to repeat ourselves as often as we did with the older participants, '*I'm sorry, but you will have to repeat this thing to me again. This young boy might have got it, but I'm still figuring it out*'. [P8, 67]

4.2.1. Commentary

Some of our participants' difficulties, especially cognitive-related ones, reinforce the results of (Guo, 2017), while others, such as those in understanding what a compiler error message is really asking them to do, were not reported in the two studies of programming and older people identified in our literature review (Guo, 2017; Ohashi et al., 2020). Most of the difficulties experienced by our participants and those in (Guo, 2017) were cognitive-related. No vision-related barrier was either observed or reported. This reinforces the relevance of cognition over vision found in the interaction barriers experienced by older people to use web technologies (Sayago and Blat, 2009) in computer programming too.

Most of our participants' learning difficulties might be due to a lack of previous experience of programming. Yet, this does not mean that they are not important, or that their importance is relative, as are just typical strains all or most beginner programmers go through, and, with experience, overcome, or get used to them. These learning difficulties had a strong, negative impact on the participants' struggle for independence, social inclusion, and feelings of incompetence - (…) '*this might be too much for us. We will never be like others, who can do these things*'. If we take into account the profile of the participants, and the results presented in 4.1, these three aspects are important parts of the participants' identity. Comments such as "like fish out of water", "not like a fool", and "we will never be like others", indicate that the experiences of the participants are often a product of the intersection (Schlesinger et al., 2017) of (old) age, immigrant, and social exclusion / inclusion, competence, and independence (i.e., not relying on anyone else). We have shown how this identity determines what programming means for the participants in Section 4.1 and in the relevance of the difficulties they faced in this section. We discuss how this identity impacts on their attitudes towards some programming environments and the way of introducing these tools to them in Section 4.5.

Previous research has shown that one of the many challenges novice programmers face from the time they write their first program are syntax-based errors and inadequate compiler error messages (Kelleher and Pausch, 2005; Becker et al., 2019). Our results reinforce this aspect by showing that it is not just older people who experience these difficulties, younger adults do too. Our results also add weight to the need for making online content more inclusive and accessible (Yesilada and Harper, 2019); in particular, online resources for learning programming, which do not seem to have attracted much research attention in the web accessibility community.[10]

Previous research has highlighted the linguistic demands of computer programming for people new to English (Guo, 2018; Pal, 2016; Dasgupta and Mako, 2017; Vogel, 2020). Most of these studies have been conducted with young students, overlooking older people. Our results show a number of linguistic demands in the programming experiences of our participants. They took extensive notes, in which they wrote down the meaning of keywords (in English) in programming that were new for them. They also developed new practices, especially the older ones, to translate and attempt to understand error messages in English without relying on anyone else. This stresses the fact that "for CS education to be for all, the field must also come to understand students' language practices" (Vogel, 2020, p. 2). However, no participant complained about English-only programming languages, as opposed to some young students in (Vogel, 2020, p.266). This might be due to their different profiles. Yet, participants acknowledged that the language issue is a barrier, especially after using Scratch and App Inventor in Spanish, as we discuss in Section 4.5.

Although older people are a very heterogeneous user group, due mostly to aging, which, as well as being a biological attribute, intertwines with a range of social dimensions (Barbosa and Vetere, 2019), our results show that our participants, and those in (Guo, 2017), experienced a number of fairly similar programming learning difficulties, regardless of their age. This suggests that older people might not be such a heterogeneous group as far as their experiences of learning programming are concerned.

Our older participants' difficulties in moving blocks in Scratch and App Inventor challenge a general perception that blocks-based programming is easier for novice programmers (Weintrop and Wilensky, 2015). Older participants learning new practices from younger ones in the sessions also challenges widespread stereotyped (mostly negative) views of older people, such as being unable to learn (Durick et al., 2013).

4.3. From disempowerment to empowerment: connecting coding with their lives, ' I feel I can do cool things now'

The first month of Course A was devoted to key programming aspects, such as conditional and iterative statements, and to do practical exercises, such as writing programs printing in the console the multiplication table of a number introduced by the user. At the beginning of the second month, the participants were much less interested and enthusiastic than they were in the first sessions. They did not talk or smile much. They seemed bored and to be running out of energy. They did not see the connection with their interests and lives:

"*I think we'll never be able to learn the basics of programing. This is* ***very difficult****. We'll need ten more courses like this"* [P6, 59]

"Yeah, I agree. I think we need to see more of a ***connection between the course and our interests****, things that are familiar to us, or our everyday lives*". [P8, 67]

Participants nodded in agreement. We then asked them to tell us things or ideas for programs in which they were interested. They suggested quite a few ideas, ranging from writing a program for calculating how many days and hours a person has lived – this was a particular curiosity of one of the participants - to creating a program with which they could learn something. From that session on, we re-designed the sessions by conducting more authentic exercises for them and making the connection between programs and their interests stronger.

In Course A, writing the code of the program that calculates how many days and hours a person has lived gave rise to a sort of competition amongst the participants. They wanted to see who had lived more hours! Participants wrote this program in pairs and compared their answers. They also solved Parsons Problems by figuring out the correct order of some lines of the program (e.g., calculating the number of hours without asking first the user to tell us his or her age) and predicting the results. We wrote these problems on the whiteboard. They did so in pairs and discussed enthusiastically the answers. Participants smiled and talked so loud (unconsciously) that other older people and members of the staff of AG came by to know '**what was going on**'. "*This is the kind of programs we want to code!*" [P9, 59] "*Absolutely, it's such fun!*" [P4, 40] Participants had exactly the same difficulties as they did in previous sessions; yet, they were more motivated to overcome them.

Participants in Courses B and C did not make, in our opinion, claims related to disempowerment or strengthening the connection of the programs made in the sessions with their lives or interests. Instead, our

[10] We read the title and abstract of the papers published in the last five years (2015-2020) in ACM-W4A and ACM-ASSETS, two key conferences in accessibility, assistive technology, and digital inclusion. We did not find any paper examining the accessibility of online resources for learning programming.

observations and conversations with them revealed that they had fun playing the Pong game in Scratch, showed creativity when drawing relatively complex pictures in Processing, and a lot of interest in creating their own app with App Inventor

"*When I was a kid, my dad told me that I should be as independent as possible. Since then, I have attempted to do things on my own. This app is really nice because I don't need to ask for help.* ***I showed it to some friends*** *yesterday and they couldn't believe I had programmed it. I can't believe I did it either,* ***I feel I can do cool things now!***" [P15, 69]

4.3.1. Commentary

Both the principles of (Guzdial, 2016) and the core elements of older people ICT learning of (Sayago et al., 2013) need a lot of detail when putting them into practice. We found it difficult to anticipate what our participants truly wanted to do and the fit of programming in their lives. Consequently, Course A was a challenge. Yet, thanks to our participants, we were able to re-design the course, adapting (Guzdial, 2016) and (Sayago et al., 2013) to the study and the courses, the context, and our participants. Doing so resulted in a shift from disempowerment to empowerment, '*I feel I can do cool things now*'.

The efficacy of tips for teaching programming for all (Brown and Wilson, 2018) with older people with low levels of educational attainment was also uncertain, due to the scant research on computer programming with this user group. Our results show that pair-based programming, live coding, the Parsons Problems, authentic tasks, and worked examples were effective and stimulating activities in the courses.

4.4. Learning to read and write but not to think in abstract terms: 'you need a brilliant mind'

By the end of the courses, we noticed that our participants had learned some elements of programming. They were able to write lines of code, such as conditional statements, without checking their notes. They also understood the logic / flow of the programs we provided them with, i.e. they were able to read and understand them. Participants' comments confirmed our observations:

"*I think we've finally* ***understood*** *why we're doing these things in this order. First you need to get the number, then obviously you need to do the math, and you need to check if the number is greater than…got it! Everything makes sense now! This is how calculators work, amazing! Learning all this only took us three months, though (smiling)*" [P7, 50].

"*Yeah, do you remember I was rather pessimistic at the beginning of the course? Now I feel more…you know,* ***I feel I am closer to this world of computers*** *and new technologies. This programming is like a new language, it can open you doors!*" [P12, 68]

Yet, the majority of the participants found it very difficult to write a program from scratch without our support, "*We don't know* ***where to begin. We are lost***" [P4, 40]. The most difficult part for all the participants was abstraction when they were faced with a new problem, "*We have this exercise. We understand it. But how do we write a program that solves it?"* [P15, 69] *"My friend, you need* ***a brilliant mind*** *to do so…and it's not mine"* [P16, 45].

4.4.1. Commentary

Learning to program is hard. In light of the results presented above, there are reasons to think that it might be even harder for non-English speaking adult people with low levels of formal education, and impossible for older people. Age-related changes in cognition make learning a challenge in older adulthood (Sayago et al., 2013). Yet, the results show that our participants actually learned some elements of computer programming, i.e., to read and write basic code, and felt '*closer to this world of computers and new technologies*'. This shows that the approach adopted in the courses, and the determination of the participants, were instrumental in achieving, in light of the difficulties they experienced, a noteworthy result. Yet, they were not able to program on their own. An important challenge, and opportunity, is how to enable them to develop the skills and abilities (e.g., abstraction) needed so that they can create the programs they want to code on their own. To this end, the results indicate that it is important to adopt a user and learner-centered design, i.e., designing for the user and their task / learning goals (Guzdial, 2016; Sayago et al., 2013), go beyond stereotyped views of older people and digital technologies (Durick et al., 2013), and acknowledge that 'professional' programmers (Halvorson, 2020), computational (data) scientists (Rao et al., 2018), makers (Kafai and Burke, 2014), or visual artists (Li et al., 2020), among others, are important, yet not the only, identities worth supporting in computer programming for all.

4.5. To block or not to block? 'Stupid or quite something!' It comes down to dealing with identity

In Course A, participants programmed in Java. At the end of the course, we introduced them to Scratch (in Spanish) by asking them to try out some of the programs in the 'explore' section of its website.[11] The objective was to gather their opinions about and attitudes towards it. The faces of the participants spoke volumes of their opinions, which highlighted social inclusion and fear of looking stupid:

"*This might be interesting for children, but not for us. I'd say that we need something simpler than the Net something… but this tool is…professional, I mean, it seems serious stuff, authentic, if you know what I mean. The Scratch one makes us seem* ***stupid****– which we are (smiling), in light of our countless difficulties in programming! - with all these colors and things"* [P5, 60]

Interestingly, this strong rejection did not happen in Course B or C, where participants interacted with textual and block-based programming languages. On the contrary, both Scratch and App Inventor were regarded as fairly (i) easy to use, "*It is easy to use in the sense that you don't write*" [P17, 45], *"I think I speak on behalf of all of us when I say that we must admit that learning in Spanish (or Catalan), is a great thing. Programming is hard, and programming in English is even harder, because you don't know the words"* [P15, 69], and (ii) useful, "*It allows you to create apps for your phones. I mean, this is complex stuff that we can do by moving blocks. That's* ***quite something!***" [P18, 63]

4.5.1. Commentary

A recent (2018) systematic review of programming with young students points out that "it is *clear* that visual programming languages present *many* benefits over traditional text-based programming languages (emphasis ours)" (Noone and Mooney, 2018). Our results present a different perspective. On the one hand, having introduced Scratch near the end of Course A, when participants had already interacted with a tool that was perceived to be for 'professionals', determined to a great extent their refusal. On the other hand, when participants experienced a number of different programming languages, Scratch was not rejected. This shows that how and when visual programming languages are introduced in courses with a certain profile of participants, who are often regarded as 'the others' in society (Riddell and Watson, 2014), matters, and that the presumed benefits of block-based programming might not always be so clear.

We do not consider that the language issue had a strong impact on the participants' perspective of Scratch. As stated above, participants did not complain about programming languages or environments being in English-only or in other languages, although they admitted, as one might expect, that Spanish was easier for them than English. The reasons for setting Scratch in Catalan were related to personal preferences, and this resonates with some of the reasons for young students to set Scratch in different languages (Vogel, 2020). From the reactions of the participants, the refusal stemmed from a clash with their identity in the courses. After having spent 3 months in a course programming with a tool targeted at professional software development, the visually

[11] https://scratch.mit.edu/ Retrieved October 13, 2020

appealing nature of Scratch made the process of learning programming appear toy-like and inauthentic (DiSalvo, 2014). This was at odds with what the participants are, or aspire to be (and to be regarded by others): socially included and competent citizens. This strong rejection did not happen in Courses B and C because the participants were introduced to different programming realities and tools.

5. Some implications

As stated in the Introduction, one of the objectives of this exploratory study is to inform future studies. To this end, we present next a number of implications that can be drawn from the results. These implications are related to understanding better older people as technology users, co-creating useful instructional materials, and designing better (more inclusive) tools for programming.

5.1. For better understanding older people as technology users

Prior works have explored differences and similarities in technology use by older and younger people, e.g. (Trewin et al., 2012; Kurniawan et al., 2019). The overall objective is to better understand older people as technology users, as older people are not well understood yet (Sayago, 2019). By working with older and younger adults in the study, we are able to address (partially) the question of how different older and younger people are as far as their programming learning experiences are concerned.

Syntax and run-time errors are two of the most important difficulties in learning programming by young novice programmers (Becker et al., 2019). These difficulties were also very important for our participants, regardless of their age, and those who participated in (Guo, 2017). Although programming has traditionally been regarded as a logical and rational activity, prior works have shown that the emotional component is very important in learning computer programming amongst students (Lishinski et al., 2017; Kinnunen and Simon, 2010). The emotional component of programming did not receive much research attention in (Guo, 2017; Ohashi et al., 2020). Our results, however, highlight how emotional learning programming was for all our participants. Connecting coding with their lives and interests, whatever these might be, from brushing up on multiplication tables to creating an app that converts Euros to another currency, was instrumental in empowering our participants. In keeping with previous research (Trewin et al., 2012; Kurniawan et al., 2019), the main difference between younger and older participants in our study was that the former were 'faster learners' than the latter. We did not witness any other remarkable differences among our participants.

Taken together, these results *suggest* that older and younger people are not so different in terms of their first experiences of programming learning as they are in other contexts, such as work environments (Fisk et al., 2009). This can be taken as an opportunity to design more inclusive experiences of programming learning, as we discuss further in Section 5.3.

5.2. For co-creating useful, user and learner-centered instructional materials in different languages and formats

Instructional materials, such as tutorials, code examples, videos, and how-to guides, play an important role in programming initiatives (e.g. *Code Week, All You Need is Code*, the *Hour of Code*), and in research (Pal and Iyer, 2015), aimed at fostering computer programming for all. However, the results show that some online materials were not useful enough for our participants, while the notes they took in the courses was their strategy for helping them remember and conduct tasks. This usefulness of their notes can be accounted for by the fact that instructions for low literate people should be as close as possible actual instances of the tasks (Medhi Thies, 2014). In this sense, turning their notes into digital ones could help us create more useful online instructional materials for this user group, and deal with linguistic demands (e.g., meaning of keywords and error messages in English) in a user, learner-centered way. Literate programming, which combines code with visualization and text in a single document, presents us with an interesting opportunity to do so.

In addition to the participants' own notes, Section 4.2 shows the relevance of the instructional approach. Participants paid a lot of attention to what we said and the programs we wrote on the whiteboard, and pointed out that "we can't learn programming by reading tutorials. We need a teacher". This suggests that the co-creation and use of video-based instructional materials could empower older and adult people to learn computer programming at their own pace. We did not explore it because in our case study it made little sense, as it is difficult to conceive of a group of older and adult people who attend physically to a course watching a video in class rather than interacting with their course mates and instructor. Yet, videos might be especially valuable for those who find it difficult to attend physically to courses, due to, for instance, age-related changes in mobility, situations of social distancing and isolation (e.g. COVID-19) (Morrow-Howell et al., 2020), or a lack of programming activities in their local area. Future research could explore the effectiveness of classroom recorded video tutorials and screencasts (a type of educational video that is created by recording the computer screen with the activities of computer screen) in programming learning with older and adult non-English speakers with low levels of formal education, and in so doing extend previous research on this issue, conducted mostly with young students (Pal, 2016; Dasgupta and Hill, 2017), making it more intergenerational.

5.3. For designing better tools for programming: avoid 'othering' them and aim for the tasks they want to do

In light of the reported learning frustrations of the respondents, (Guo, 2017) addresses the important question of how to design better programming tools for older people. This question is aligned with the most widespread design approach in age-targeted learning programming, wherein there is a tendency to design specific tools for particular user groups (Kafai and Burke, 2014; Kelleher and Pausch, 2005), and in HCI research with older people too (Sayago, 2019).

On the one hand, our results reinforce the need for better designs. Our results challenge a general perception that block-based programming is easier for novice programmers (Weintrop and Wilensky, 2015), due to the participants' (mostly older people) difficulties in dealing with the mouse. Our results also show that error and notification messages should be improved to prove more informative for our participants. As opposed to (Denny et al., 2014), wherein it is argued that enhancing error messages appears ineffectual because undergraduate students did not read the messages, our participants did read them. Future studies could explore the extent to which principles for designing error messages (Becker et al., 2019) apply to older and adult people with low levels of formal education. Doing so could contribute to address a key, and yet-to-be-addressed, question in research on programming errors: what does a good programming error message look like? (Becker et al., 2019) Our results also show that some design solutions adopted in programming tools, such as highlighting incorrect code, clashed with the practices our participants have developed as a result of using word document editors. Alternative ways of bringing incorrect code to the attention of the users (e.g., using audio feedback or altering the size of the word or line with the code error) could be explored. Our results also indicate that the emotional component of the programming experiences of our participants was important, and designing more emotional-sensitive programming tools could provide them with richer programming experiences. Future research could explore the design of programming tools that recognize the emotions of the programmers, or make them more aware of their emotional statuses, and adapt the user interface according to them.

One the other hand, our results do not support the idea of designing

tools specifically for older people. The (lack of) rejection of Scratch in the courses indicates that more important than age was the identity of our participants and how the tools and the way we introduced them projected it. When Scratch was introduced after having used ordinary tools, Scratch was rejected because it was perceived 'not for them'. However, when programming tools such as Jupyter Notebooks and Netbeans, which could have also been regarded as 'not for them', because these are used by people with a different profile (e.g., data scientists and professional programmers), were introduced as tools that people use to learn to program, and do their jobs, participants did not reject them. Our participants struggle for independence and social inclusion. 'Othering' (Riddell and Watson, 2014) them, i.e. building specific tools for them, does not seem to be the best strategy for achieving their goals. Instead, it should be possible to build programming environments (and, perhaps, languages) for the tasks the participants want, without "othering" them. For example, Weintrop et al. (2018) provide empirical basis for the use of block-based programming, designed for young novices, as an effective programming interface for the growing set of applications and contexts where programming by non-experts (e.g. adult novices engaged in industrial robot programming) might occur.

6. Limitations

A limitation of case studies is that their results as not so generalizable as those gathered in other research methods (Lazar et al., 2017). The findings of this case study, with a particular profile of older, and adult people, might be even less generalizable. However, case studies are close examinations that can be used to build understanding, present evidence for the existence of certain behavior, or to provide insight that would otherwise be difficult to gather. This case study has revealed and explained the experiences of programming learning of a group of older and adult people that are unlikely to be gathered by adopting other methods, for instance, online surveys.

Our review of previous and related works on computer programming and older people focused on papers published in English and available at three large academic databases. This can be seen as a limitation, as there might be studies published in other languages, and in other academic databases, that are not discussed in this paper. Future studies, perhaps systematic reviews, could improve this issue.

We have discussed a number of methodological aspects throughout Section 3. The challenge for us was how (and whether) programming could be introduced to our participants, and their programming learning experiences be explored. This challenge is visible in the mistakes we made, especially in the first course, and the relevance of the lessons learned from this first experience to carry out the other courses. We do not claim that other approaches, programming languages or environments, would not have enabled us to explore the participants' programming learning. We do not claim that our own background has not affected the study either. Yet, the approach we adopted enabled us to provide new evidence of older people and computer programming, learn how not to do things, identify relevant aspects of their first programming experiences, conduct courses that were useful for the participants, and draw some implications for research and design.

As stated in the introduction, the maker movement represents a significant push in the coder movement. We considered exploring programming and making by addressing Arduino, and kits such as MakeyMakey[12] with Scratch, in the courses. Yet, as the study developed, we realized that exploring programming within the context of making with our participants was another case study, and future research can explore it.

We do not claim that other, or even different, implications can be drawn from this study, as the readers can interpret the results in different ways. We have discussed those implications that, in our view, are important, grounded in the results, and can spark considerable future research. The implications might also be regarded as general or not too detailed. Since our aim was to spark future research, we decided to highlight important issues and leave the details of their exploration and implementation up to future studies.

We have not explored computer programming over time (i.e., extended periods of time). We decided to explore our participants' first experiences of programming because we considered that their initial encounters could help us identify important issues in their relationship with programming, and inform future research studies that can strengthen it.

7. Conclusion

We began this paper by arguing that exploring computer programming with older people is timely and worthwhile. However, the scant research on this arena, along with widespread (and mostly) negative stereotyped views of older people and digital technologies, can lead us to believe that programming 'is not for older people', especially those with low levels of formal education, and be skeptical about the feasibility of exploring their programming experiences, and the contributions this exploration, if possible at all, can make to Human-Computer Interaction. Yet, this paper shows otherwise.

This paper has presented an exploratory case study aimed to examine the computer programming learning experiences of a group of older and adult active computers users with low levels of formal education and no previous experience of programming. Over a 6-month period, we provided a hands-on introduction to several textual and visual programming languages and environments to 29 participants in three courses, wherein they were engaged in a number of different programming tasks, some more engaging than others. By the end of the courses, participants reported, and our observations confirmed, that they were able to learn, understand, and write simple programs. Although this case study is not enough to conclude that older people create more accessible and useful technologies for themselves by learning programming, the results challenge stereotypes, contribute to the progression that positions older people as producers of digital content by adding new evidence to it, and show that programming can empower them and strengthen their perceived social inclusion.

Based on first-hand observations and conversations with the study's participants, this paper has shown relevant factors that shape, and help us understand, their computer learning experiences. We have addressed their motivations for learning programming, which include feeling more socially included and competent, and learning more about how computers work. We have shown the most important difficulties the participants experienced, most of them cognitive-related. We have identified the meaning these difficulties had for them in terms of their identity, key aspirations and struggle for independence and social inclusion, and the role these aspects take on in how and why programming languages and environments might be rejected or adopted. We have highlighted the importance of connecting coding with their lives to empower them. We have argued that older and younger participants experienced remarkably similar first programming learning experiences. These results deepen and widen current understanding of older people and digital technologies, and research on computer programming for all, as we have discussed throughout the paper.

Although the results of this case study might be difficult to generalize, they contribute to Human-Computer Interaction. Given that programming with older people is a mostly unexplored research arena, an important objective of this case study was to inform future studies. Consequently, the case study was explorative, and we have presented implications for research and design that can be drawn from the results. These implications address a range of aspects, from understanding better older people as technology users, co-creating useful instructional materials, and designing better (more inclusive) tools for programming. As we have discussed, these implications help us explore and understand

[12] https://makeymakey.com/ Retrieved October 13, 2020

better the relationship between older people and computer programming, and digital technologies, in general. Future studies can confirm or reject it.

In our future work, we aim to explore the comment made by one our participants further, 'This programming is like a new language'. In particular, and inspired by Coding as Another Language (Bers, 2019), we want to understand whether and how programming can be taught as another language to older and adult people, and the effectiveness of doing so. We plan to work together with the adult educators in AG and other centers to design the activities. We also aim to explore literate programming by having our participants to write their notes (in their own language) in an online tool designed for the programming tasks they want to do and use them in courses. We also aim to explore web programming, e.g. Javascript (and HTML + CSS), as it can be of interest to the participants (e.g., web apps), the design of more useful error and notification messages, along with the emotional component of the programming experiences of older and adult people, by conducting further courses and running co-design sessions.

CRediT authorship contribution statement

Sergio Sayago: Conceptualization, Methodology, Formal analysis, Investigation, Writing - original draft, Supervision, Funding acquisition. **Ángel Bergantiños:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing - review & editing.

Declaration of Competing Interest

None

Acknowledgements

We are deeply thankful to AG participants, who helped us learn a lot and carry out this study, and who were always willing to share their programming experiences with us as well as allowing us to share them with the rest of the world. Thanks to our colleagues Mireia Ribera and Josep Blat for useful discussions, to Paula Forbes for improving our English and our ideas. We also acknowledge the support from the Barcelona City Council through the AGORA 4.0 project. We also thank the reviewers of this manuscript for their detailed, rigorous, and insightful comments and suggestions.

References

Barbosa, S., Markopoulos, P., Pattern, F., Stumpf, S., Valtolina, S (Eds.), 2017. End-User Development. Springer.
Barbosa Neves, B., Vetere, F., 2019. Ageing and Digital Technology Designing and Evaluating Emerging Technologies For Older Adults. Springer, Berlin Heidelberg.
Baudisch, P., Mueller, S., 2017. Personal Fabrication. Found. Trends® Human–Computer Interact 10, 165–293. https://doi.org/10.1561/1100000055.
Becker, B.A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D.J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.M., Pearce, J.L., Prather, J., 2019. Compiler error messages considered unhelpful: the landscape of text-based programming error message research. In: Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, pp. 177–210. https://doi.org/10.1145/3344429.3372508.
Bergström, I., Blackwell, A.F., 2016. The practices of programming. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, pp. 190–198. https://doi.org/10.1109/VLHCC.2016.7739684.
Bers, M.U., 2019. Coding as another language: a pedagogical approach for teaching computer science in early childhood. J. Comput. Educ. 6, 499–528. https://doi.org/10.1007/s40692-019-00147-3.
Braun, V., Clarke, V., 2019. Reflecting on reflexive thematic analysis. Qual. Res. Sport. Exerc. Heal. 11, 589–597. https://doi.org/10.1080/2159676X.2019.1628806.
Brewer, R., Piper, A.M., 2016. Tell it like it really is": a case of online content creation and sharing among older adult bloggers. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. San Jose, California, USA, pp. 5529–5542. https://doi.org/10.1145/2858036.2858379.
Brewer, R., Ringel, M., Piper, A., 2016. Why would anybody do this? older adults' understanding of and experiences with crowd work. In: CHI.
Brown, N., Wilson, G., 2018. Ten quick tips for teaching programming. PLoS Comput. Biol. 14 (4), 1–8.
Coffey, A., 1999. The Ethnographic Self. Fieldwork and the Representation of Identity. SAGE Publications.
Cunningham, K., 2018. The novice programmer needs a plan. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC. IEEE, pp. 269–270. https://doi.org/10.1109/VLHCC.2018.8506481.
Dasgupta, S., Hill, B.M., 2017. Learning to code in localized programming languages. In: L@S 2017 - Proceedings of the 4th (2017) ACM Conference on Learning at Scale, pp. 33–39. https://doi.org/10.1145/3051457.3051464.
Denny, P., Luxton-Reilly, A., Carpenter, D., 2014. Enhancing syntax error messages appears ineffectual. In: ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference, pp. 273–278. https://doi.org/10.1145/2591708.2591748.
DeWalt, K., DeWalt, B., 2011. Participant observation. A guide For Fieldworkers. Altamira Press, New York.
Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A (Eds.), 2015. End-User Development. Springer.
Disalvo, B., 2014. Graphical qualities of educational technology: using drag-and-drop and text-based programs for introductory computer science. IEEE Comput. Graph. Appl. 34, 12–15. https://doi.org/10.1109/MCG.2014.112.
Durick, J., Brereton, M., Vetere, F., Nansen, B., 2013. Dispelling ageing myths in technology design,. In: OzCHI. Adelaide, Australia, pp. 467–476.
Ferreira, S.M., Sayago, S., Blat, J., 2017. Older people's production and appropriation of digital videos: an ethnographic study. Behav. Inf. Technol. 36 https://doi.org/10.1080/0144929X.2016.1265150.
Fisk, A., Rogers, W., Charness, N., Czaja, S.J., Sharit, J. (Eds.), 2009. Designing For Older Adults. Principles and Creative Human Factors Approaches. CRC Press.
Guo, P.J., 2018. Non-native english speakers learning computer programming: barriers, desires, and design opportunities. In: CHI p. Paper 396.
Guo, P.J., 2017. Older adults learning computer programming: motivations, frustrations, and design opportunities. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 7070–7083. https://doi.org/10.1145/3025453.3025945.
Gupta, D., 2004. What is a good first programming language? XRDS 10, 4 (August 2004), 7. https://doi.org/10.1145/1027313.1027320.
Guzdial, M., 2016. Learner-Centered Design of Computing Education. Morgan & Claypool. Synthesis Lectures on Human-Centered Informatics.
Halvorson, M.J., 2020. Code Nation. Personal Computing and the Learning to Program Movement in America. ACM Books.
Hammersley, M., Atkinson, P., 2007. Ethnography. Principles in Practice. Routedge, London (UK).
Jelen, B., Monsey, S., Siek, K.A., 2019. Older adults as makers of custom electronics. In: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19, pp. 1–6. https://doi.org/10.1145/3290607.3312755.
Kafai, Y.B., Burke, Q., 2014. Connected code. Why children Need to Learn Programming. The MIT Press.
Kelleher, C., Pausch, R., 2005. Lowering the Barriers to Programming: a survey of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 83–137. https://doi.org/10.1145/1089733.1089734.
Kinnunen, P., Simon, B., 2010. Experiencing programming assignments in CS1: the emotional toll. In: ICER'10 - Proceedings of the International Computing Education Research Workshop, pp. 77–85. https://doi.org/10.1145/1839594.1839609.
Kitchen, R., Dodge, M., 2011. Code / Space. Software and Everyday Life. The MIT Press.
Knowles, M., Holton, E., Swanson, R., 2005. The Adult learner. The definitive Classic in Adult Education and Human Resource Development. Elsevier, Burlington, MA.
Kurniawan, S., Arch, A., Smith, S., 2019. Ageing and Older Adults. In: Yesilada, Y., Harper, S. (Eds.), Web Accessibility. A Foundation For Research. Springer Human-Computer Interaction Series, p. 2019.
Lazar, J., Feng, J.H., Hochheiser, H., 2017. Research methods in human-computer interaction. Res. Methods Human-Comput. Interact. https://doi.org/10.1016/b978-044481862-1/50075-3.
Li, J., Brandt, J., Mech, R., Agrawala, M., Jacobs, J., 2020. Supporting visual artists in programming through direct inspection and control of program execution. In: CHI. Honolulu, pp. 1–12. https://doi.org/10.1145/3313831.3376765.
Lishinski, A., 2017. Students' emotional reactions to programming projects in introduction to programming: measurement approach and influence on learning outcomes. In: Icer '17. https://doi.org/10.475/123.
Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C.J., Burnett, M.M., 2016. Programming, problem solving, and self-awareness: effects of explicit guidance. In: Conference on Human Factors in Computing Systems - Proceedings, pp. 1449–1461. https://doi.org/10.1145/2858036.2858252.
Malizia, A., Valtolina, S., Morch, A., Serrano, A., Stratton, A., 2019. End-User Development. Springer.
Medhi Thies, I., 2014. User interface design for low-literate and novice users: past, present and future. found. Trends® Human–Comput. Interact. 8, 1–72. https://doi.org/10.1561/1100000047.
Meissner, J.L., Vines, J., McLaughlin, J., Nappey, T., Maksimova, J., Wright, P., 2017. Do-it-yourself empowerment as experienced by novice makers with disabilities. In: Proceedings of the 2017 Conference on Designing Interactive Systems - DIS '17., pp. 1053–1065. https://doi.org/10.1145/3064663.3064674.
Mohorovičić, S., Strčić, V., 2011. An overview of computer programming teaching methods. In: Proc. 22nd Cent. Eur. Conf. Inf. Intell. Syst., pp. 47–52.
Morrow-Howell, N., Galucia, N., Swinford, E., 2020. Recovering from the COVID-19 pandemic: a focus on older adults. J Aging Soc Policy 32 (4–5), 526–535. https://doi.org/10.1080/08959420.2020.1759758.
Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., Zander, C., 2008. Debugging: the good, the bad, and the quirky -a qualitative analysis of novices'

strategies. In: SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education, pp. 163–167. https://doi.org/10.1145/1352135.1352191.

Newell, A.F., 2011. Design and the digital divide. insights from 40 years in computer support for older and disabled people. Morgan & Claypool. Synthesis Lectures on Assistive, Rehabilitative, and Health-Preserving Technologies.

Noone, M., Mooney, A., 2018. Visual and textual programming languages: a systematic review of the literature. J. Comput. Educ. 5, 149–174. https://doi.org/10.1007/s40692-018-0101-5.

Ohashi, Y., Yamachi, H., Murokoshi, Y., Kumeno, F., Tsujimura, Y., 2020. Development of a programming course for senior citizens taught by senior citizens. In: ICIET. Japan, pp. 18–23.

Ortega García, A., Ruiz-Martínez, A., Valencia-García, R., 2017. Using App Inventor for creating apps to support m-learning experiences: a case study. Comput. Appl. Eng. Educ. https://doi.org/10.1002/cae.21895.

Pal, Y., 2016. A Framework for Scaffolding to Teach Programming to Vernacular Medium. Indian Institute of Technology Bombay.

Pal, Y., Iyer, S., 2015. Classroom versus screencast for native language learners: effect of medium of instruction on knowledge of programming. In: Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, pp. 290–295. https://doi.org/10.1145/2729094.2742618.

Pang, A., Anslow, C., Noble, J., 2018. What programming languages do developers use? a theory of static vs dynamic language choice. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing. VL/HCC. IEEE, pp. 239–247. https://doi.org/10.1109/VLHCC.2018.8506534.

PEW. 2014. Older adults and technology use. Retrieved from https://www.pewresearch.org/internet/2014/04/03/older-adults-and-technology-use/ July 13, 2020.

Rao, A., Bihani, A., Nair, M., 2018. Milo: a visual programming environment for data science education. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing. VL/HCC. IEEE, pp. 211–215. https://doi.org/10.1109/VLHCC.2018.8506504.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J.A.Y., Silverman, B., Kafai, Y., 2009. Scratch: programming for all. Commun. ACM 52, 60–67. https://doi.org/10.1145/1592761.1592779.

Riddell, S., Watson, N. (Eds.), 2014. Disability, Culture and Identity. Routledge.

Rosales, A., Fernández-ardèvol, M., Comunello, F., Mulargia, S., Ferran-ferrer, N., 2017. Older people and smartwatches, initial experiences. El Profesional de La Información 26 (3), 457–463.

Rushkoff, D., 2010. Program Or Be programmed. Ten Commands For a Digital Age. OR Books, New York.

Sánchez Aroca, M., 1999. La Verneda-Sant Martí: a school where people dare to dream. Harv. Educ. Rev. 69 (3), 320–335.

Sayago, S. (Ed.), 2019. Perspectives On Human-Computer Interaction Research with Older People. Springer Human-Computer Interaction Series.

Sayago, S., Blat, J., 2009. About the relevance of accessibility barriers in the everyday interactions of older people with the web. In: W4A 2009 - International Cross Disciplinary Conference on Web Accessibility.

Sayago, S., Forbes, P., Blat, J., 2013. Older people becoming successful ICT learners over time: challenges and strategies through an ethnographical lens. Educ. Gerontol. 39 https://doi.org/10.1080/03601277.2012.703583.

Sayago, S., Rosales, A., Righi, V., Ferreira, S.M., Coleman, G.W., Blat, J., 2016. On the conceptualization, design, and evaluation of appealing, meaningful, and playable digital games for older people. Games Cult. 11 https://doi.org/10.1177/1555412015597108.

Schehl, B., Leukel, J., Sugumaran, V., 2019. Understanding differentiated internet use in older adults: a study of informational, social, and instrumental online activities. Comput. Human Behav. 97, 222–230. https://doi.org/10.1016/j.chb.2019.03.031.

Schlesinger, A., Edwards, W.K., Grinter, R.E., 2017. Intersectional HCI: engaging identity through gender, race, and class. In: Proc. 2017 CHI Conf. Hum. Factors Comput. Syst. (CHI '17), pp. 5412–5427. https://doi.org/10.1145/3025453.3025766.

Schneider, H., Eiband, M., Ullrich, D., Butz, A., 2018. Empowerment in HCI - a survey and framework. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18, pp. 1–14. https://doi.org/10.1145/3173574.3173818.

Sim, T.Y., Lau, S.L., 2019. Online tools to support novice programming: a systematic review. In: 2018 IEEE Conference on E-Learning, e-Management and e-Services, IC3e, 2018. IEEE, pp. 91–96. https://doi.org/10.1109/IC3e.2018.8632649.

Tanenbaum, J.G., Williams, A.M., Desjardins, A., Tanenbaum, K., 2013. Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2603–2612. https://doi.org/10.1145/2470654.2481360.

TIOBE. https://www.tiobe.com/tiobe-index/ Retrieved July 13, 2020.

Trewin, S., Richards, J.T., Hanson, V.L., Sloan, D., John, B.E., Swart, C., Thomas, J.C., 2012. Understanding the role of age and fluid intelligence in information search. In: ASSETS. Boulder, Colorado, pp. 119–126.

Vee, A., 2017. Coding literacy. How computing Programming is Changing Writing. The MIT Press, Cambridge (USA).

Vines, J., Pritchard, G., Wright, P., Olivier, P., 2015. An age-old problem: examining the discourses of ageing in HCI and strategies for future research. ACM Trans. Comput. Interact. 22, 1–27.

Vogel, S., 2020. Translanguaging About, With, and Through Code and Computing: Emergent Bi / Multilingual Middle Schoolers Forging Computational Literacies. The City University of New York.

Weintrop, D., Wilensky, U., 2015. To block or not to block, that is the question: students' perceptions of blocks-based programming. In: Proceedings of IDC 2015: The 14th International Conference on Interaction Design and Children, pp. 199–208. https://doi.org/10.1145/2771839.2771860.

Weintrop, D., Wilensky, U., 2015b. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In: ICER 2015 - Proceedings of the 2015 ACM Conference on International Computing Education Research, pp. 101–110. https://doi.org/10.1145/2787622.2787721.

Weintrop, D., Afzal, A., Salac, J., Francis, P., Li, B., Shepherd, D.C., Franklin, D., 2018. Evaluating CoBlox: a comparative study of robotics programming environments for adult novices. Conf. Human Factors Comput. Syst. - Proc. 1–12. https://doi.org/10.1145/3173574.3173940.

Weintrop, D., Holbert, N., 2017. From blocks to text and back: programming patterns in a dual-modality environment. In: Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, pp. 633–638. https://doi.org/10.1145/3017680.3017707.

Wolber, D., Abelson, H., Friedman, M., 2014. Democratizing computing with app inventor. GetMobile 18, 53–58.

Yesilada, Y., Harper, S. (Eds.), 2019. Web Accessibility. A Foundation For Research. Springer Human-Computer Interaction Series.