CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Visualisering av referenser i Python

Utvecklandet av LearnyPy, en webbapplikation för att främja undervisningen av programmering

Kandidatarbete inom data- och informationsteknik

Moa Berglund
Saga Hassellöf
Simon Johansson
Alex Phu
Vera Svensson
Yenan Wang

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

Kandidatarbete 2022

Visualisering av referenser i Python

Utvecklandet av LearnyPy, en webbapplikation för att främja undervisningen av programmering

Moa Berglund
Saga Hassellöf
Simon Johansson
Alex Phu
Vera Svensson
Yenan Wang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg, Sverige 2022

Visualisering av referenser i Python
Utvecklandet av LearnyPy, en webbapplikation för att främja undervisningen av programmering
MOA BERGLUND SAGA HASSELLÖF SIMON JOHANSSON ALEX PHU VERA SVENSSON YENAN WANG

Handledare: Krasimir Angelov, institutionen för data- och informationsteknik
Examinator: Magnus Myreen, institutionen för data- och informationsteknik

Kandidatarbete 2022
Institutionen för data- och informationsteknik
Chalmers tekniska högskola och Göteborgs universitet
SE-412 96 Göteborg
Telefon: +46 31 772 1000

Omslag: Bilden visar LearnyPys logotyp, som består av namnet ”LearnyPy” tillsammans med två pilar och texten ”Visualize Python”.

Typsatt i LaTeX
Göteborg, Sverige 2022

Abstract

Understanding how variables and objects are connected can be difficult for novice programmers. The need for educated programmers is growing and effective tools to teach programming are therefore desirable. This project aims to provide a tool to simplify the procedure of teaching references in Python to new programmers. The target group for using the tool is both those new to programming and those teaching it. The tool should be able to handle user input containing Python code and illustrate how each line of code affects the references. The intention with this illustration is to contribute to a better understanding of the results from executing the code.

The resulting product, LearnyPy, is a website containing a code editor where the user can enter Python code they want to visualize. LearnyPy also has a toolbar that allows the user to execute the whole code all at once or in steps. The execution of the code is handled by Skulpt, an in-browser implementation of Python. For each execution the current state is visualized by a graph, using the library d3-graphviz. The graph contains all variables and objects as nodes and the references are shown as directed edges. A terminal is also provided to show output and possible error messages. Because LearnyPy is implemented as a website it is accessible to the public. The user interface of the website is built using the JavaScript library React. The project is conducted with the intention to support further work on the codebase. The result shows that the project meets the purpose to a large extent. There is opportunity for improvement and further work is encouraged regarding the user experience.

Sammanfattning

Att förstå hur variabler och objekt relaterar till varandra kan vara svårt för nybörjare inom programmering. Behovet av utbildade programmerare växer och effektiva redskap för att undervisa programmering är därför önskvärt. Syftet med detta projekt är att tillhandahålla ett redskap som förenklar processen att undervisa referenser i Python till nya programmerare. Den tänkta målgruppen för verktyget är både de nya inom programmering och de som lär ut det. Verktyget ska kunna hantera indata, i form av Pythonkod, från användare och illustrera hur varje kodrad påverkar referenserna. Avsikten med denna illustration är att bidra till en bättre förståelse för resultaten från exekvering av koden.

Den resulterande produkten, LearnyPy, är en webbplats som innehåller en kodeditor där användaren kan mata in Pythonkod de vill visualisera. LearnyPy har också ett menyfält som låter användaren exekvera hela koden i ett svep eller i steg. Exekveringen av koden hanteras av Skulpt, en webbaserad implementering av Python. För varje exekvering visualiseras det aktuella tillståndet med en graf, hanterat av biblioteket d3-graphviz. Grafen innehåller alla variabler och objekt som noder och referenser som riktade kanter. En terminal finns också för att visa utdata och eventuella felmeddelanden. Då LearnyPy är implementerat som en webbplats är den tillgänglig för allmänheten. Användargränssnittet för webbplatsen är byggt med JavaScript-biblioteket React. Projektet har genomförts med intentionen att stödja ytterligare utveckling av kodbasen. Resultatet visar är att projektet till stor del uppfyller syftet. Det finns utvecklingspotential och förslag till vidare arbete för att förbättra användarupplevelsen.

Nyckelord: undervisning, Python, Skulpt, referens, programmering, webbapplikation, graf, visualisering

Förord

Vi vill tacka vår handledare Krasimir Angelov som har erbjudit oss hjälp och rådgivning under kandidatarbetet. Vi riktar även ett tack till de som tog sig tid att delta i projektets användartester.

Innehåll

Figurer

Tabeller

1 Introduktion

I takt med att dagens samhälle digitaliseras alltmer, ökar även behovet av kunskap kring programmering. Detta speglas av att ämnet programmering är en del av läroplanen sedan 2018 för grundskolor i Sverige [1]. Samtidigt är det många privatpersoner som av olika anledningar väljer att själva utforska programmering. Några motiveringar till detta är att allt fler arbeten kräver grundläggande programmeringskunskaper och att trenden pekar på att programmering i olika former styr alltmer av gemene mans vardag [2]. Som en följd av behovet och intresseökningen för programmering erbjuds allt fler eftergymnasiala utbildningar rörande ämnet. Vad som däremot begränsar denna motivation hos både samhälle och människa, är att programmering vanligen upplevs ha en hög inlärningströskel [3].

För det första kan tröskeln till förståelse för programmering upplevas hög, då det inte bara handlar om att kunna skriva regelmässig kod som utför en specifik uppgift. Inte sällan innebär starten av programmeringsutbildning även att den blivande programmeraren behöver lära sig hantera en mängd olika programvaror som behöver laddas ned och initieras korrekt [4]. För det andra innebär att behärska programmering även en förståelse för vad som händer bakom kod-raderna under exekveringen. Detta är något som inte är helt enkelt att undervisa [5] och är problemet som detta projekt i huvudsak kommer att behandla.

1.1 Bakgrund

Uppdragsgivaren [6] till detta projekt, som undervisar programmering på Chalmers Tekniska Högskola (vidare benämnt som Chalmers), berättar att han i nuläget undervisar vad som händer med variabler och objekt genom att relationer mellan dem modelleras på en tavla. Detta är resurskrävande då en kunnig person behöver finnas tillgänglig för att måla upp scenariot för varje student som vill ta del av förklaringen visuellt. Samtidigt begränsar det studenten från att studera självständigt. Vad som däremot är positivt med denna undervisningsform är den visuella delen. Enligt forskning ökar modellering tempot på inlärning, vilket gör det till en av de mest effektiva metoderna för undervisning [7]. När studenter får något konkret att associera kunskapen till blir det även enklare att förankra och diskutera den.

Nedan följer ett scenario för att sätta in läsaren i hur en glimt av inlärningsfasen av programmeringskoncept kan se ut.

Föreställ dig att du ska lära dig ditt första programmeringsspråk, Python. Tidigare har du fått lära dig om variabler och hur man tilldelar värden till dessa. Denna föreläsning ska ni lära er om listor. Föreläsaren skriver följande på tavlan:

```
x = [1, 2, 3]
y = x
y.append(4)
print(y)
print(x)
```

Föreläsaren frågar sedan "Vad kommer att skrivas ut?". Du tittar på dina anteckningar från föregående föreläsning och hittar följande:

```
a = 1
b = a
b = 2
print(b)
print(a)
```

Under har du antecknat att *2* följt av *1* skrivs ut. Detta beror på att när b tilldelas värdet *2* så kommer endast b påverkas av denna ändring. Du vet att både x och y, från dagens exempel, har tilldelats samma lista. Du är dock osäker på om y kommer tilldelas en annan lista, på ett liknande sätt som för heltalen, när något ändras i listan. Det visar sig att *[1, 2, 3, 4]* skrivs ut för både x och y, trots att det inte var x som utökades. Detta beror på att vissa av Pythons objekt är oföränderliga (eng. immutable), medan andra är föränderliga (eng. mutable) [8]. Effekterna av detta och varför det kan uppfattas som förvirrande för nybörjare inom programmering, kommer att diskuteras i denna rapport. Rapporten kommer presentera utvecklingen och resultatet av LearnyPy, ett redskap för inlärning av hur referenser mellan variabler och objekt fungerar i Python.

1.2 Syfte

Syftet med detta projekt är att utveckla en webbaserad applikation, vid namn LearnyPy, som kan visualisera relationer mellan variabler och objekt i Python. Intentionen med LearnyPy är att den ska kunna användas som ett hjälpmedel för inlärning av programmering genom att hjälpa studenter att förstå referenser till objekt. För att alla som vill utnyttja programmet för sitt lärande ska kunna göra det, är målet att LearnyPy ska vara tillgängligt för allmänheten.

1.3 Frågeställningar

- Hur kan relationer mellan variabler och objekt i Pythonkod visualiseras på ett pedagogiskt sätt?
- Hur ska applikationen göras tillgänglig för allmänheten?
- Hur kan applikationen skapas på ett sätt som är enkelt för andra utvecklare att arbeta vidare på?

1.4 Avgränsningar

Projektets avgränsningar delas upp i två kategorier: de relaterade till slutprodukten och de relaterade till utvecklingsprocessen. Först kommer utvecklingsprocessens avgränsningar presenteras, och senare slutproduktens avgränsningar.

1.4.1 Utveckling

I utvecklandet av LearnyPy eftersträvades att använda färdiga verktyg, för att påskynda och underlätta utvecklingen. I enighet med detta avgränsades projektet till att inte skapa en egen kompilator, då detta inte var av tillräckligt värde med tanke på utvecklingstiden. Detta ledde dock till ytterligare avgränsningar, såsom att LearnyPy inte kan behandla all Pythons funktionalitet i och med att kompilatorn som integrerades inte stödjer det.

1.4.2 Slutprodukt

LearnyPy ska enbart tillämpas som en webbapplikation, alltså inget plug-in, ingen mobilapplikation eller liknande. Dessutom är fokus i första hand att skapa en användbar, snarare än en visuellt avancerad, produkt. Ambitionen är ett modernt och användarvänligt gränssnitt, eftersom det krävs för att LearnyPy ska bli använd och omtyckt. Mer än så är dock inte av prioritet så länge det fanns mer funktionalitet att utveckla i LearnyPy.

Målgruppen för LearnyPy är nybörjarprogrammerare, vilket LearnyPys kapacitet avgränsas utifrån. LearnyPy behandlar endast simpla stycken Pythonkod, om cirka 10-20 rader. Detta medför att visualiseringen inte behöver vara anpassad för att kunna visa mer variabler och objekt samtidigt än vad som ryms på dessa rader.

Ytterligare en avgränsning är att LearnyPy inte behöver stödja andra visualiseringar än den som projektet skapar. Python-paket som ger stöd för olika visuella komponenter går alltså inte att tillämpa i LearnyPy. Detta för att webbsidan inte ska ha för mycket som händer på skärmen samtidigt och på så vis göra den svårare att använda.

2
Teori

I detta kapitel förklaras relevanta koncept och verktyg som bidrar till förståelse av LearnyPy och besluten som har tagits under utvecklingen. Detta kapitel kommer endast avhandla programmeringskoncept som rör projektet och antar en viss förståelse av programmering hos läsaren.

2.1 Generell minneshantering

Minnet hos ett datorprogram består simplifierat vanligen av tre olika delar: text, statiskt minne och dynamiskt minne [9]. Textsektionen innehåller maskinkod av programmet. Statiskt minne innehåller de variabler som endast deallokeras efter programmet har terminerat, såsom statiska variabler. Till skillnad från statiskt minne innehåller dynamiskt minne variabler som skapas och deallokeras dynamiskt under programmets exekvering. För att hantera dynamiskt skapade variabler används antingen en stack eller en heap [9], beroende på hur tilldelning till variabeln sker.

Stacken kan liknas med en stapel av disk [10]. Det går att lägga till saker högst upp på stacken, och när något tas bort från stacken är det vad som ligger högst upp som tas bort. Heapen är ett mindre strukturerat minne, där objekt kan förvaras mer utspritt (beroende på implementationen) och har en bestämd minnesplats [11]. Stacken har fördelen att vara lite snabbare [12], då det endast krävs en enda operation för att ta bort någonting. Toppen av stacken ligger vanligen i CPUn [13], vilket gör den lättillgänglig. Anledningen till varför allting inte lagras på stacken är för att det är svårt att ändra på föremål placerade där. Om disk-liknelsen ska användas igen så är det svårt att byta ut en tallrik som redan ligger mitt i diskhögen.

2.2 Python

Python är ett imperativt programmeringsspråk vars syfte är att vara enkelt att utveckla program med och enkelt att lära sig [14]. En av Pythons implementationer kallas CPython [15]. Python kommer vara synonymt med CPython i denna rapport. Först kommer detta avsnitt förklara vad som menas med konceptet ”namn” i Python. Sedan kommer de datatyper som är föränderliga och oföränderliga tas upp samt vad detta innebär. Till sist kommer referenser i Python behandlas för att ge förståelse för vilka situationer som kan upplevas förvirrande för nybörjarprogrammeraren.

2.2.1 Definition av variabler i Python

Centralt i Python, till skillnad från vissa andra programmeringsspråk, är att istället för att skapa variabler som innehåller eller pekar på ett visst värde, finns det ”namn”

som pekar på objekt [16]. Ett namn är vad en typisk programmerare hade kallat för en variabel, eftersom de båda fungerar på samma sätt syntaktiskt. När en variabel tilldelas ett värde i Python skapas ett namn som refererar till ett visst objekt, oavsett datatyp. Variabler kommer därför att vara synonymt med namn i detta projekt. Alla dessa objekt kommer även att sparas på heapen enligt [17], vilket innebär att Python endast använder stacken för att hålla koll på funktionsanrop [18].

2.2.2 Föränderliga och oföränderliga objekt

Objekt i Python kan vara antingen föränderliga (eng. mutable) eller oföränderliga (eng. immutable). Tabell 2.1 visar några av Pythons inbyggda typer och om de är föränderliga eller oföränderliga.

Tabell 2.1: Några vanliga datatyper i Python kopplat till om de tillåter att förändra objekt eller inte [19].

Typ	Föränderliga
int	nej
float	nej
double	nej
bool	nej
string	nej
tuple	nej
list	ja
set	ja
dict	ja

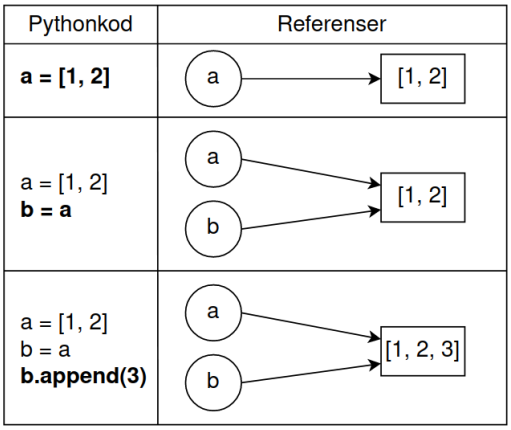
”Nej” i tabellen innebär att ett objekt med denna datatyp är oföränderligt. Detta innebär att om exempelvis en variabel refererar till detta objekt och en modifiering görs via denna variabel, kommer variabeln att tilldelas en referens till ett annat objekt innehållande resultatet av modifieringen. ”Ja” i tabellen innebär att ett objekt av denna datatyp är föränderligt. Om en förändring sker via exempelvis en variabel som refererar till ett föränderligt objekt, kommer själva objektet modifieras och variabeln behåller samma referens som innan ändringen.

2.2.3 Referenser i Python

Referenser är ett sätt att koppla en variabel till en datamängd, och på så sätt veta vart i minnet denna datamängden är placerad [20]. Tänk att programmeraren har en stor datamängd som ska skickas till en funktion. Det är då dyrt att kopiera hela datamängden till funktionen, eftersom det tar upp dubbelt så mycket plats i minnet. Då kan istället referensen skickas till funktionen, vilket gör att funktionen kan ta reda på vart den datamängden är placerad i minnet och undviker därav en onödig kopia.

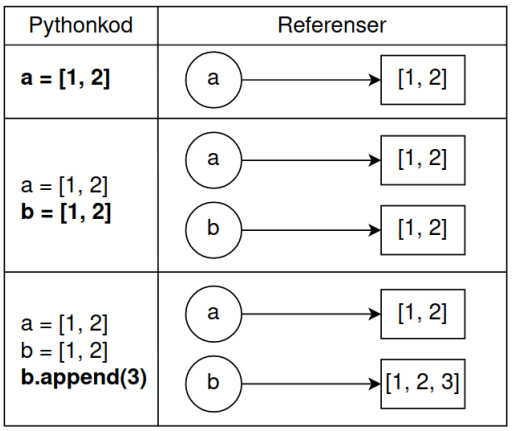
Användningen av referenser kan alltså vara praktisk, men kan leda till oförutsedda sidoeffekter om inte programmeraren är medveten om vilka variabler som refererar till samma objekt, och om objektet i sin tur är föränderligt eller inte. Detta är något

som vanligen är en stor utmaning för nybörjare inom programmering [5]. Se Figur 2.1 för en illustration av hur problematiken kan uppstå. Figuren visar att variabel *a* refererar till en lista. I nästa steg skapas variabel *b* som tilldelas *a*. Det kallas att *b* är alias för *a* och vice versa [21], då referensen som *a* innehåller nu även sparas i *b*. När en förändring av listan sker via *b* i sista steget, så innebär det även en förändring av objektet som *a* refererar till. Om inte hänsyn tas till att både *a* och *b* refererar till samma föränderliga objekt så utgör denna förändring en oförutsedd sidoeffekt.



Figur 2.1: Exempel på när problematik kring alias kan uppstå. När *3* läggs till i listan via variabel *b* påverkas även *a*, då *a* och *b* har referens till samma objekt.

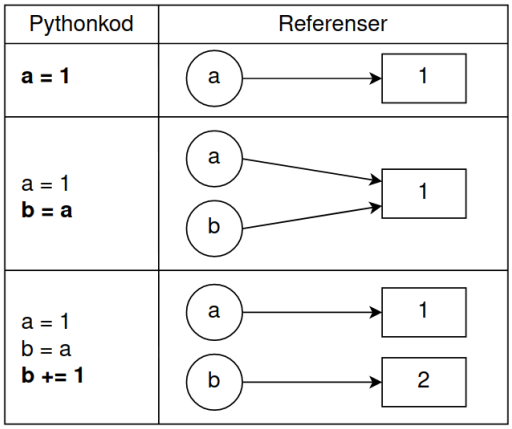
Som visats i 2.1 kommer en modifiering av ett föränderligt objekt medföra att alla variabler som refererar till detta objekt påverkas. I figur 2.2 visas hur referenserna ser ut om *b* istället skulle tilldelas ett nyskapat objekt med exakt samma innehåll som *a*. Figuren visar att detta skulle leda till att *a* och *b* refererar till olika objekt. En modifiering av det nyskapade objektet skulle här inte leda till en förändring av det objekt som *a* refererar till.



Figur 2.2: Exempel på när problematik kring alias inte uppstår. Då *b* har tilldelats ett nyskapat objekt kommer *b* att referera till ett annat objekt i minnet än *a* trots att *a* och *b* tillsynes innehåller samma värde. När *3* läggs till i listan via variabel *b* påverkas inte *a*, då *a* refererar till ett annat objekt.

Problematiken med alias gäller endast för objekt som är föränderliga. Om exemplet i figur 2.1 istället skulle hantera oföränderliga objekt skulle inte samma sidoeffekter

uppstå. Se exempelvis figur 2.3 där a och b tilldelas samma oföränderliga objekt, ett heltal. Trots att de två variablerna refererar till samma objekt kommer inte en modifiering, som exempelvis en ökning av heltalet, via variabel b leda till en förändring för a.



Figur 2.3: Exempel på modifiering av variabler som refererar till oföränderliga objekt. Trots att a och b refererar till samma objekt så kommer inte a påverkas av den modifiering som sker via b. Istället kommer modifieringen leda till att b tilldelas en referens till ett objekt innehållandes resultatet.

2.3 Skulpt

Skulpt är en webbaserad implementering av Python [22]. Skulpts implementation av Python eftersträvar att fungera som CPython [22] och har därför många liknelser med CPython, men viss funktionalitet saknas. Användaren kan med Skulpt skriva, kompilera och exekvera Pythonkod direkt i webbläsaren utan att ladda ner något externt. Skulpt körs lokalt i webbläsaren och kräver således ingen server för att kompilera och exekvera koden. Detta tack vare att Skulpt är utvecklat i JavaScript. Skulpt har en utvecklargemenskap som tillåter att alla intresserade utvecklare kan föreslå tillägg till projektet från separata grenar [23], vilket ger Skulpt stora möjligheter att växa. Förslagen är öppna för alla att se och bygga vidare på i sin tur.

2.4 Webbutveckling, JavaScript och React

För att kunna köra Skulpt krävs en webbplats att integrera allt i. Webbsidor är egentligen inget mer än en text i HTML kombinerat med CSS, som webbläsaren i sin tur tolkar till det som användaren ser. HTML och CSS i sig är kraftigt statiska, vilket betyder att det är svårt att ändra innehållet på en webbsida när den väl har laddats in i användarens webbläsare. Detta kan dock lösas genom användningen av programmeringsspråket JavaScript, som har förmågan att ändra strukturen och innehållet av dokumentet på webbsidor.

Det finns många olika sätt att utveckla en webbplats, varav ett är att använda biblioteket React. React är skrivet i JavaScript, och har som syfte att utveckla

webbsidor med design och funktionalitet i åtanke. Den grundläggande premissen med React är att det möjliggör att programmera dynamiska webbplatser med traditionell programmeringsstruktur. Med hjälp av React går det att göra uträkningar och operationer i JavaScript, för att sedan returnera en modifierad HTML-fil.

Som tidigare nämnt så är det fullt möjligt att skapa dynamiska webbplatser med enbart HTML, CSS och JavaScript. Däremot kan utvecklingen av sådana sidor underlättas genom användningen av React eftersom detta bibliotek hanterar majoriteten av den underliggande logiken för en webbplats. Detta leder till att utvecklaren kan fokusera på skapandet av användargränssnittet, istället för lågnivålogiken som till exempel hur renderingen av visuella komponenter sker.

2.5 Testning i React

Programvarutestning är en central del inom mjukvaruutveckling för att garantera korrekthet av slutprodukten och förenkla utvecklingsprocessen. Genom testning av programvaran kan kvaliteten och säkerheten förbättras eftersom felen, med större sannolikhet, upptäcks tidigt. Därav minskar risken för kritiska programfel i en senare utvecklingsfas. En godtycklig testtäckning främjar även utvecklingsprocessen, eftersom ändringar i kodbasen snabbt kan kontrolleras för korrekthet med de färdigställda testerna.

Det finns åtskilliga testningsramverk för JavaScript. Ett modernt React-projekt har redan två testningsramverk inbyggt [24]: Jest och React Testing Library. Jest är ett ramverk med det primära syftet att exekvera testfall och hantera testsviter i en virtuell miljö som simulerar en webbläsare [25]. Till skillnad från Jest hanterar React Testing Library inga enskilda tester, men det erbjuder en mängd av hjälpmetoder på högnivå för att testa React-komponenterna [25], vilket förenklar skrivprocessen för testerna.

2.6 Netlify

För att göra en webbapplikation tillgänglig för allmänheten så är det relevant att webbapplikationen kan nås på nätet. Detta kan åstadkommas genom att webbapplikationen hyses själv med hjälp av en server eller genom användingen av ett webbhotell som agerar som värd för webbplatser. Netlify [26] är ett av många webbhotell. Utöver att vara ett webbhotell så har Netlify även en smidig integrering med andra populära webbtjänster där öppen källkod kan förvaras, som t.ex. GitHub. Netlify automatiserar då processen av att bygga och sedan utplacera webbapplikationen när det skett förändringar i källkoden.

2.7 Tillgänglighet för webbplatser

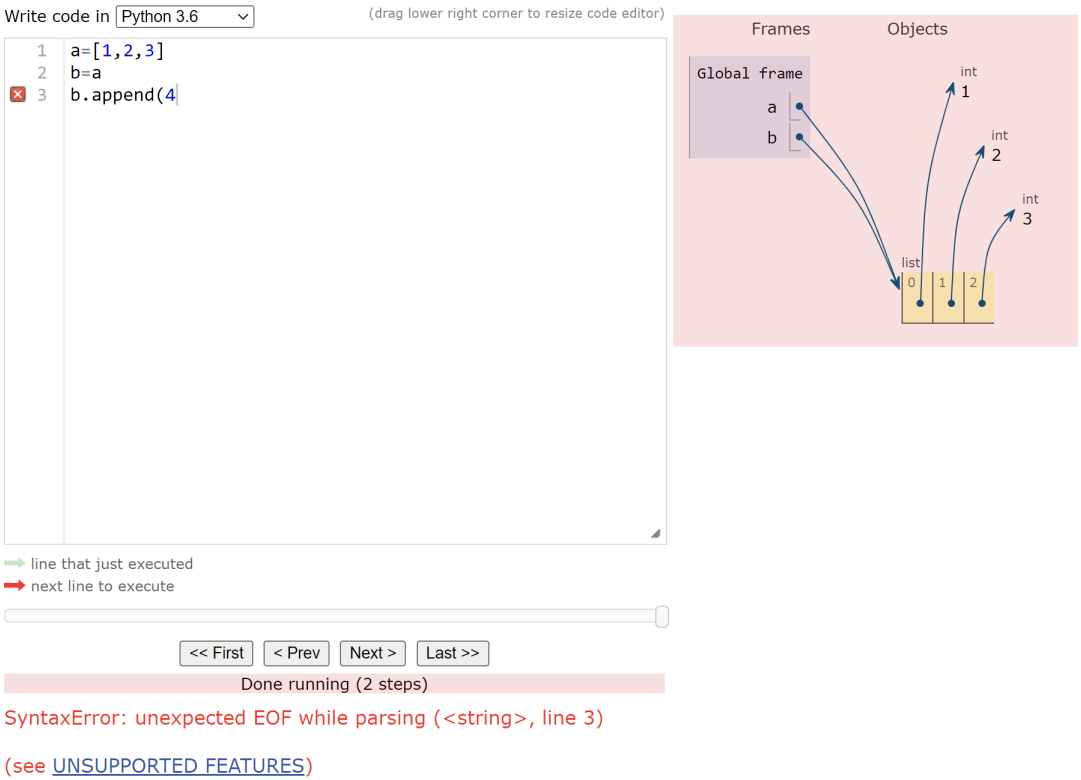
15 % av den globala populationen är drabbad av någon typ av funktionsnedsättning, och det är en växande siffra [27]. All datorinteraktion medför ett dilemma kring den etiska aspekten av exkludering på grund av olika funktionsnedsättningar. Exempel på funktionsnedsättningar som kan begränsa förmågan att använda en dator är olika typer av synfel (såsom defekt färgseende och blindhet) och olika faktorer som begränsar användandet av en datormus (reumatism, förlamningsskador, darrhänthet) [28]. Förenta Nationerna förmedlar i konventionen om mänskliga rättigheter för personer med funktionsnedsättning att anslutna länder ska se till att människor med funktionsnedsättningar kan delta till fullo på internet, och vid tillämpning av information- och kommunikationstekniker [29].

Myndigheten för digital förvaltning, vanligen kallad DIGG, kommunicerar kraven för tillgänglighet på webbplatser och mobila applikationer som gäller för offentliga aktörer i Sverige [30]. Sedan december 2016 är det lag på att i huvudsak uppfylla tre punkter [31]. Den första är att följa den europeiska standarden EN 301 549 V3.2.1 [32] som är baserad på den internationella standarden WCAG 2.1 [33] till minst nivå AA. Den andra är att användare ska ha möjlighet att ge återkoppling kring huruvida riktlinjerna uppfylls. Det sista är att webbplatser och mobila applikationer för offentliga aktörer i Sverige ska innehålla en tillgänglighetsredogörelse. En tillgänglighetsredogörelse förmedlar information kring hur riktlinjerna är uppfyllda och hur användaren kan återkoppla till utgivare eller ansvarig. Tillgänglighetsredogörelsen ska även innehålla en länk till DIGG, för att användaren ska kunna anmäla brister gällande tillgängligheten hos den besökta webbplatsen eller applikationen.

3
Nulägesanalys

Eftersom projektet syftar till att skapa en webbapplikation som främjar inlärning av programmering, är det relevant att utforska liknande produkter. I detta avsnitt kommer därför Python Tutor analyseras för att hitta dess för- och nackdelar, vilka har bidragit med inspiration till LearnyPy. Python Tutor [34] är ett verktyg, som likt detta projekt, syftar till att hjälpa användaren att förstå grundläggande koncept inom programmering genom att visualisera vad som sker i minnet under kodexekveringen. Detta verktyg utvecklades ursprungligen av Philip Guo [35] men har senare vidareutvecklats av andra, tack vare den öppna källkoden.

Det finns två val för programmering med visualisering i Python Tutor. Det första alternativet är att koden visualiseras direkt när den skrivs, se figur 3.1. I detta alternativ blir det en liten fördröjning mellan skrivandet av kod och uppdaterad visualisering. Som figuren visar så syns även felmeddelanden om användaren inte skriver fullständig kod. Nybörjarprogrammerare kan tänkas bli avskräckta av att se felmeddelanden då de inte garanterat förstår att många av felen uppstår för att koden inte är färdigskriven. För att göra LearnyPy så enkel och intuitiv som möjligt att använda för den tänkta målgruppen erbjuds inte denna direkta visualisering.



Figur 3.1: Omgående uppdatering av visualisering av kod på webbplatsen Python Tutor [34]. Felmeddelandet syns under kodrutan, samt att visualiseringen är röd för att indikera att något är fel i koden.

Det andra alternativet för programmering med visualisering i Python Tutor är att användaren skriver koden i kodrutan för att sedan visualisera koden. I figur 3.2 visas gränssnittet för detta alternativ. I detta fall går det inte att redigera koden när den är visualiserad. För att förändra koden krävs att användaren klickar på ”Edit this code”, som syns under kod-sektionen i figuren nedan. Användaren leds då vidare till den ursprungliga sidan där koden är möjlig att redigera.

Figur 3.2: Visualisering av kod efter att den är färdigskriven på webbplatsen Python Tutor [34].

Till skillnad från redigering av kod under exekvering, upplevs detta som ett enklare alternativ för nybörjarprogrammeraren att förhålla sig till. Därav erbjuder LearnyPy ett liknande exekveringsalternativ som Python Tutors alternativ där koden låses under exekvering. Dock kräver inte LearnyPy, till skillnad från Python Tutor, att användaren växlar mellan olika sidor för redigering och visualisering av koden. Istället styr användaren när koden ska visualiseras, samt när visualiseringen ska avbrytas så att koden kan redigeras. För att tydliggöra att koden inte kan redigeras under visualisering låses kodrutan för ändringar och antar en skuggad design. LearnyPys funktionalitet och gränssnitt presenteras mer ingående i avsnitt 5.2.

Som avslutande analys noteras att Python Tutor är serverbaserad [35]. Att låta en applikation vara serverbaserad kan potentiellt exponera applikation för olika risker och prestandarelaterade problem. Därför håller LearnyPy serverberoendet så lågt som möjligt. Detta motiveras vidare i avsnitt 4.1.1.

4
Metod

I detta kapitel presenteras de metoder som användes under projektets gång, samt motiveringar för varför just dessa valdes. Metoderna kategoriseras i tre delar: Integrerad programvara, Utvecklingsprocess och Användartest.

4.1 Val av integrerad programvara

I enighet med projektets avgränsningar användes flera programvaror för att effektivare göra framsteg. I detta avsnitt lyfts fem programvaror som spelar en avgörande roll för projektets resultat. Den första är Skulpt, som möjliggör att applikationen är webbaserad och inte serverdriven. För att använda data från Skulpt i webbläsaren dynamiskt användes React, som också underlättade skapandet av ett interaktivt användargränssnitt. Visualiseringen av variabler och objekt gjordes med en riktad graf från biblioteket Graphviz [36], eftersom det är ett tydligt och brus-reducerat sätt att representera en mängd noder (variabler och objekt) och kanter (relationer). Den fjärde centrala integrerade programvaran är tjänsten Netlify, som agerar värd för webbplatsen. Sist användes biblioteket CodeMirror [37] för att skapa en kodeditor. De fem nämnda programvarorna presenteras mer ingående nedan, tillsammans med deras fördelar och eventuella nackdelar.

4.1.1 Kompilator

Skulpt valdes som grund för att kompilera Pythonkod. Ett argument till varför Skulpt valdes är för att det möjliggör att återanvända mycket färdig funktionalitet. En annan strategi hade istället kunnat vara att skapa en egen Python interpretator. Med den tid och kunskap som fanns tillgänglig för projektet hade interpretatorn endast kunnat hantera en liten del av Pythons funktionalitet. För att möjliggöra att detta projekt kan förbättras av efterföljande arbeten så är det av vikt att projektet ska vara skalbart. Detta skulle försvåras om det även innefattade att utöka interpretatorn. Förutom dessa fördelar med att integrera Skulpt i projektet, får applikationen dessutom tillgång till de utökningar som görs till Skulpt. Dessvärre medför detta att projektets resultat skulle kunna bli negativt påverkat av att Skulpt blir utökat. Om något som detta projekt beror på blir förändrat i Skulpt kan det innebära att LearnyPy inte längre fungerar som önskat, men denna nackdel ansågs vara mindre än fördelarna med att integrera Skulpt.

Skulpt kommer även med fördelen att kodexekvering kan genomföras direkt i webbläsaren. Detta möjliggör mindre kommunikation mellan klienten och servern eftersom Skulpt tillhandahåller all information som LearnyPy kan behöva. Detta betyder alltså att den enda interaktion som sker mellan klient och server är när användaren besöker webbplatsen för projektets webbapplikation, det vill säga när användaren laddar ner alla filer som krävs för att kunna visa webbapplikationen på sin enhet.

Anledningen till att hålla serverberoendet så lågt som möjligt är för att en serverdriven applikation förlorar prestanda desto fler användare som använder sidan [38]. Detta kan göra att applikationen upplevas långsammare och mindre lättanvänd, vilket inte är önskvärt i ett visuellt interaktivt program.

Fördelen är dessutom att servern inte riskerar några injektionsattacker [39]. En injektionsattack är en attack där användaren inför kod som kan vara skadlig för servern. Ett exempel på hur en sådan attack kan ske är om det saknas särkerhetscheckar. Då kan en användare exempelvis skriva in kod som hämtar information om andra användares lösenord. Eftersom LearnyPy endast exekveras i användarens enhet så går det inte att exekvera skadlig kod på servern på detta sätt.

Dessa fördelar med Skulpt sker på bekostnad av den krävande tid det tar att läsa in ett ofullständigt dokumenterat projekt till en sådan nivå att det går att använda sig av dess funktionalitet i en applikation. För att effektivisera och minska denna inläsningstid experimenterades det tidigt med Skulpt och hur det skulle kunna användas i den tänkta applikationen, för att fokusera läsandet kring just det projektet kräver. Det är värt att nämna att ett annat bibliotek upptäcktes senare i projektet som är benämnd Pyodide. Till skillnad från Skulpt så är Pyodide inte en JavaScript implementation av Python utan istället CPython som har kompilerats till WebAssembly [40]. Pyodide valdes dock inte eftersom det upptäcktes för sent under projektet, och kommer att diskuteras mer i kapitel 6.5.

4.1.2 Bibliotek för webbutveckling

React används i projektet på grund av att det blir ständigt uppdaterat av engagerade programmerare och har en lång rad olika bibliotek och verktyg tillgängligt, samt är väl integrerat med webbutveckling vilket ger många möjligheter. Då det är väl etablerat finns det mycket information på internet som kan främja processen när utvecklare stöter på motgång i programmerandet. React upplevdes även som ett tidseffektivt val eftersom flera medlemmar i projektet hade tidigare erfarenhet av detta bibliotek.

4.1.3 Bibliotek för visualisering av grafer

Biblioteket d3-graphviz valdes för att illustrera grafer över variabler, objekt och deras relationer genom att en svg-fil genereras. d3-graphviz använder sig av Graphviz, en programvara som tillhandahåller visualisering av grafer [41]. Graphviz gav möjlighet att formge grafens komponenter på ett sätt som passar projektets visioner. Nodernas utseende anpassades för att förmedla skillnaden mellan variabler och objekt i grafen. Grafen som genereras är baserad på en sträng uppbyggd av DOT-språket, ett språk uppbyggt av en abstrakt grammatik [42]. Med detta språk ges stor kontroll över hur grafen utformas. Exempelvis kan noder byggas upp av flera celler och från var cell kan kanter kopplas. Detta möjliggör att exempelvis listor kan illustreras med celler för var index i listan. Kanter kopplade till dessa celler visualiserar den referens indexet förvarar. Inledningsvis användes biblioteket react-graph-vis [43] för

visualisering av grafer. Anledningen till bytet av bibliotek från react-graph-vis till d3-graphviz var just på grund av Graphvizs stora möjligheter för utformning av noder, vilket react-graph-vis inte upplevdes ha motsvarighet till.

4.1.4 Bibliotek för kodeditor

För att göra skrivandet av kod i LearnyPy så smidig som möjligt eftersträvades en kodruta som liknar en standard editor. Istället för att göra en egen från grunden upp, användes biblioteket CodeMirror [37]. Codemirror erbjuder att skapa en kodruta med syntaxfärgning, radnummer, flertalet olika färgteman etc.

4.1.5 Webbhotell

Från början tänktes GitHub Pages [44] användas som värd för projektets webbapplikation. GitHub Pages var speciellt lockande med tanke på att källkoden för webbapplikationen redan var förvarad i GitHub. På detta vis skulle all funktionalitet finnas på en plattform. Under integreringen av GitHub Pages så uppstod det däremot problem. För att kunna använda deras tjänst så krävdes det en installation av ett paket för att göra webbapplikationen kompatibel. Problemet som uppstod var att den lokala utvecklingsmiljön inte längre fungerade med detta paket, vilket förhindrade en fortsatt utveckling av webbapplikationen. Därför valdes istället webbhotellet Netlify, vilket var enklare att integrera eftersom det inte krävdes någon installation av paket för att göra webbapplikationen kompatibel med deras tjänst. En ytterligare fördel är att en förhandsvisning av webbapplikationen fås under granskningsfasen av Netlify. Detta innebär att den som granskar ny kod inte behöver hämta den nya koden och starta upp projektet i sin lokala utvecklingsmiljö för att se de nya förändringarna.

4.2 Utvecklingsprocess

Projektet bedrevs i form av agil utveckling, framförallt då det är ett inkluderande och dynamiskt arbetssätt som möjliggjorde att varje deltagare kunde växla mellan olika områden under projektets gång. Som hjälpmedel användes en Scrumtavla för att organisera programmerandet. Bilaga A visar en instans av projektets Scrumtavla under projektets sjunde vecka.

För att få ytterligare struktur kring utvecklingen där sex personer programmerar på samma projekt så sattes några förhållningsregler upp. Den första var att alltid jobba i olika grenar, alltså inte direkt i huvudgrenen, för att undvika konflikter mellan olika utvecklares kod. För att få förflytta nytillkommen kod till kodbasen bestämdes det att minst två personer, som inte var med och skrev den nya koden, skulle gå igenom den och godkänna att koden är lämplig att införa till kodbasen. Några kriterier som kontrollerades vid dessa granskningar var att det inte fanns buggar i koden, att syntaxen följde standarden som gruppen bestämt, att lösningen upplevdes rimlig och möjlig att bygga vidare på. Dessa granskningar kunde självklart inte garantera att alla kriterier alltid uppfylldes, på grund av mänskliga faktorer och att det inte förväntades att granskningen skulle ta mer än några minuter. De fungerade

även som ett sätt för de som inte var med och programmerade den aktuella koden att ta del av vad som tillkommit på ett tydligt sätt.

4.2.1 Utvecklingskontrakt

Vid starten av utvecklingsprocessen skapades ett kontrakt för hur data som skickas mellan olika filer i projektet ska vara formaterad. Detta för att kunna arbeta i olika områden parallellt men ändå vara säker på att de kommer vara kompatibla när de sammanfogas.

4.2.2 Prototyp

För att förenkla utvecklingen av användargränssnittet skapades tidigt en prototyp av webbsidans användargränssnitt, se Bilaga B. Syftet var att på kort tid få en gemensam överblick av vilka funktioner applikationen skulle erbjuda samt att ha en översiktlig mall att utgå från vid utvecklandet av gränssnittet.

4.2.3 Kontinuerlig integrering

Då stora delar av detta projekt innefattar programmering där flera personer samarbetar tillämpades kontinuerlig integrering (CI). Sammanfattningsvis innebär CI att testning och utvärdering utförs på ny kod innan den sammanfogas till kodbasen [45]. Att introducera ett extra lager av granskning ökar chansen att upptäcka problem som kan uppstå när olika utvecklares kod kombineras. Detta medför i sin tur att problemet kan korrigeras tidigare än om problemet identifieras senare i processen. Följden av denna praktik blir att bättre kodkvalitet säkerställs.

Mer specifikt utnyttjades GitHub Workflows för att exekvera den automatiska testningen, som i detta projekt bestod av ESLint och Prettier. ESLint [46] är ett verktyg som analyserar JavaScript-kod för att hitta problem medan Prettier [47] används för att garantera att kodbasen upprätthåller en enhetlig kodstil.

4.2.4 Testning

Testerna skrevs parallellt med utvecklandet av LearnyPy och förhållningssättet var att skriva relevanta tester direkt efter implementering av ny funktionalitet. Detta för att upptäcka felen innan de kan ackumulera och framställa ett större problem. Testerna skrevs med de inbyggda testningsramverken i React. Trots att de inbyggda testningsramverken är syftade att enbart testa användargränssnittet, användes dessa även för att testa funktionaliteter som egentligen inte beror av användargränssnittet. Förklaringen till det är att inga funktionaliteter går att testa utan ett existerande användargränssnitt, då skriptet från Skulpt endast kan användas när det har överförts till HTML.

Framförallt testades korrektheten av logik som skapats för att tillhandahålla korrekta referenser mellan variabler och objekt, med syftet att kontrollera så relationen mellan variabler och objekt tolkas korrekt. För testningen av referenser innefattar

testfallen exempelvis instansiering och tilldelning av variablerna mellan olika typer. Likartat implementerades även en mängd av integrationstester. Till skillnad från funktionalitetstestning, där fokus ligger på logik, testar integrationstestningen användargränssnittet och samspelet mellan visuella komponenter på webbplatsen. Däremot testades inte alla möjliga fall där något skulle kunna misslyckas. Endast ett testfall skrevs för varje del i programmet där det kan misslyckas. Det skrevs alltså inte flera tester av samma typ med varierande parametrar eftersom uttömmande testning ansågs för tidskrävande och förmodas vara en omöjlig uppgift i webbutveckling [48].

4.3 Användarvänlighet

För att göra LearnyPy användarvänligt användes i huvudsak två strategier. Den ena var att följa standarder för hur webbplatser kan göras användarvänliga för funktionsnedsatta användare. Den andra var att testa LearnyPy i sin tänka användningsmiljö, av nybörjarprogrammerare, för att säkerhetsställa att LearnyPy är designad på ett sätt som uppmuntrar den tänkta användningen. Dessa två strategier beskrivs mer nedan.

4.3.1 Uppnå tillgänglighet

Eftersom projektet är en konsekvens av att undervisare på Chalmers efterfrågat produkten valdes det att utgå från att förhålla sig till samma tillgänglighetsnivå som Chalmers har gjort på sin officiella webbplats. Detta för att LearnyPy ska vara användbar under universitetets regim. Chalmers tillgänglighetsredogörelse [49] förmedlar att universitetet eftersträvar att uppfylla WCAG 2.1 till nivå AA, i enighet med riktlinjerna för offentliga aktörer som presenterades i avsnitt 2.7. Därav eftersträvades samma nivå under utvecklingen av LearnyPy, och det som inte gick att uppfylla av olika anledningar listades i en tillgänglighetsredogörelse, i enighet med riktlinjerna. Likt Chalmers webbplats struktur valdes det att placera redogörelsen under ett meny-alternativ som syftar till information om webbplatsen och dess organisation.

Eftersom det huvudsakliga syftet med LearnyPy är att hjälpa studenter att förstå relationer mellan variabler och objekt, är det centralt att användaren kan ta del av den genererade grafen av just detta. Därav var det prioriterat att skapa en beskrivande text för grafen, som kan läsas upp av skärmläsare, så att även de användare som inte tar del av informationen visuellt kan få nytta av LearnyPy. Texten förmedlades sedan genom att använda HTML-attributet ”aria-label”.

4.3.2 Användartest

Användartesterna genomfördes i två steg. Det första användartestet användes för att utveckla produkten i rätt riktning, medan det senare syftade till att utvärdera produkten. Det sista användartestet presenteras därför i nästa kapitel där resultatet redogörs. I varje test utförde ett fåtal användare ett bevakat test av applikationen

och svarade på ett antal frågor. Resultatet från dessa analyserades för att förbättra applikationen.

Målgruppen för användartesterna var nybörjarprogrammerare i enlighet med projektets syfte. Resultatet från användartesterna hade därför hög tillförlitlighet i och med att deltagarna i användartesterna var i princip identiska representationer av den åsyftade målgruppen för LearnyPy. Detta innebar att problem som upptäcktes under användartestningen sannolikt skulle besvära senare användare om problemen inte åtgärdades.

Deltagarna i den första omgången av användartester bestod därmed av ett fåtal nybörjarprogrammerare mellan 20 och 30 år från Sverige. En sammanfattning av de första användartesterna finns i Bilaga C. Nedan sammanfattas de åtgärder som vidtogs efter första omgången av användartester:

- Över hälften av användarna reagerade på valet av ett ljust färgtema och önskade ett mörkt färgtema istället. Därför utvecklades funktionaliteten att byta mellan ljust och mörkt tema.
- Då användarna ombads stega igenom koden, radera en del av koden och sedan fortsätta exekvera, var flertalet förvirrade över utfallet. Funktionaliteten skrevs därför om så att kodrutan låses för indata under avbuggning för att öka användarvänligheten.
- En användare var förvirrad över att ”kör”-knappen inte startade om exekveringen av koden efter ett felmeddelande visats. Detta var en bugg som åtgärdades.
- En användare hittade att for-loopar inte visualiserades. Detta var en bugg som åtgärdades.
- En användare kommenterade på att grafen borde centreras varje gång koden exekveras. Detta åtgärdades.

Det första användartestet visar att LearnyPy hjälpte några som inte innan förstod kod med typiska referens-manipulationer att faktiskt förstå förklaringen till utfallet av koden. Därav förändrades inte strukturen för visualiseringen inför nästa användartest.

5
Resultat

Nedan presenteras resultatet av metoden, alltså den slutgiltiga produkten, som går att nå på webbadressen `https://learnypy.netlify.app/`. Kapitlet börjar med en genomgång av webbplatsens gränssnitt, och fotsätter med att presentera funktionaliteten hos dess komponenter. Sedan presenteras resultatet av de slutgiltiga användartesterna, som en utvärdering kring huruvida projektet har uppfyllt sitt mål. Kodens filstruktur samt tillämpning av designmönster presenteras därefter.

5.1 Användargränssnitt

LearnyPys syfte är att vara ett läroverktyg riktat till nybörjare inom programmering. Därför finns ett värde i att produkten är designad för att underlätta användningen.

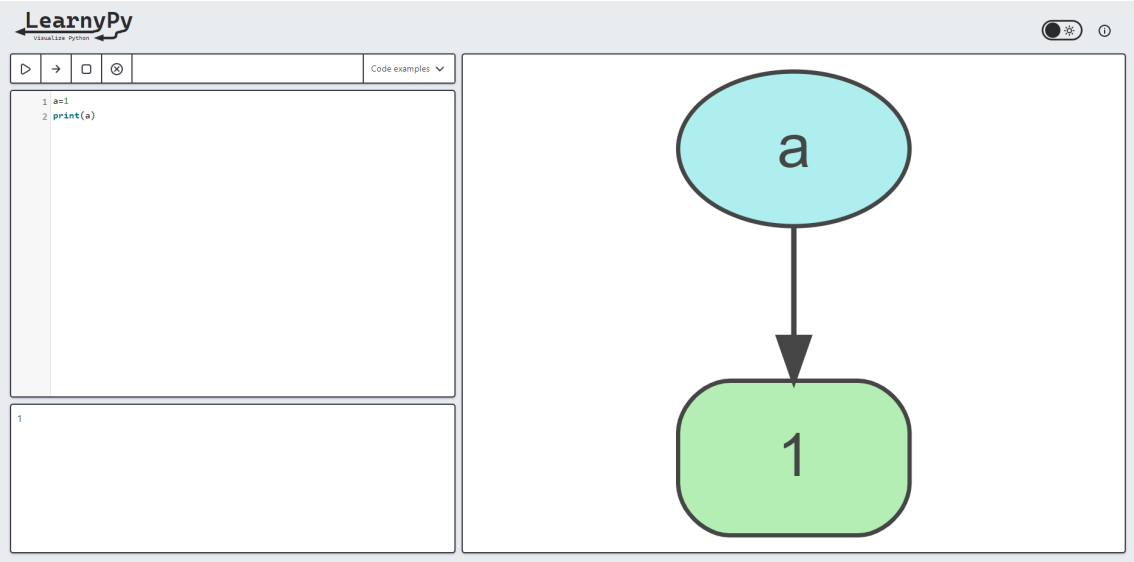
I LearnyPy används ikoner framför text på så många knappar som möjlig då detta möjliggör att förklara en funktionalitet utan att behöva kunna ett specifikt språk. Ikonerna är hämtade från Jam, en webbplats som tillgängliggör 900+ ikoner i svg-format [50]. Ambitionen var att använda välkända ikoner för knappens funktionalitet, med ett utseende som håller LearnyPy avskalad och enhetlig. Det sågs som en onödig arbetsbörda att designa egna då det inte fanns några avgörande fördelar med att göra egna. För att öka användarvänligheten används tooltips för knapparna. Dessa innehåller ett förklarande ord eller en kort förklarande text på engelska. LearnyPy stödjer enbart engelska.

Färgerna är valda i syfte att skapa ett engagerande och harmoniskt utseende. De är också anpassade efter färgblindhet genom att färgernas kontraster gentemot varandra är godkända enligt WCAG 2.1 nivå AA, med hjälp av verktyget Contrast Checker [51]. Det finns en knapp som erbjuder användaren att välja mellan ljust eller mörkt tema.

LearnyPy är skapad med responsiv webbdesign i åtanke. Detta innebär att layouten i LearnyPy automatiskt anordnar sig så att den alltid är välanpassat till skärmstorleken av användarens enhet. LearnyPy är därför även en mobilvänlig webbplats.

5.2 Genomgång av LearnyPys funktionalitet

Den slutgiltiga produkten är en webbplats där användaren kan skriva in Pythonkod, och sedan se en graf över den inskrivna kodens variabler och objekt samt deras referenser. Användargränssnittet för LearnyPy visas i figur 5.1 nedan.



Figur 5.1: Användargränssnittet hos LearnyPy. Längst uppe till vänster finns LearnyPys logotyp. Under den ligger menyfältet och under menyfältet finns kodrutan. Terminalen ligger under kodrutan. Den stora rutan till höger är visualiseringsrutan. Längst upp till höger ligger en knapp för att byta mellan olika färglägen och en knapp som leder till informationssidan.

Visualiseringen av koden sker när användaren klickar på ”kör”-knappen, förutsatt att inga fel är fångade av kompileringen. Visualiseringen är också integrerad med funktionaliteten för att stega igenom koden, och uppdateras då för varje rad. Nedan följer en genomgång av varje komponent i LearnyPy.

5.2.1 Menyfält

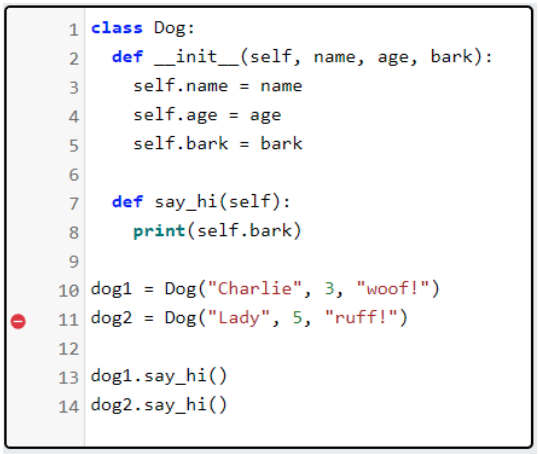
Menyfältet (figur 5.2) består av olika knappar som styr exekveringen av koden. Den första knappen från vänster gör så att koden i kodrutan exekveras i ett svep, alltså utan avbrott eller någon form av stegning. Knappen därefter gör så att den markerade raden i kodrutan exekverar, och stannar på raden därefter. Nästa knapp stannar all exekvering av programmet, och rensar både terminalen och visualiseringsrutan. Den sista knappen tar bort alla brytpunkter som är placerade i programmet.

Knappen längst till höger är en rullgardinsmeny med exempelprogram. Klickar användaren på ett exempel i menyn, fylls kodrutan med exemplet och terminalen samt visualiseringsrutan rensas. Syftet med exempel är att bidra med exempel som kan vara svåra för en nybörjarprogrammerare att förstå, i enighet med undersökningen [3] som belyser att exempelprogram är ett uppskattat sätt att lära sig programmering.

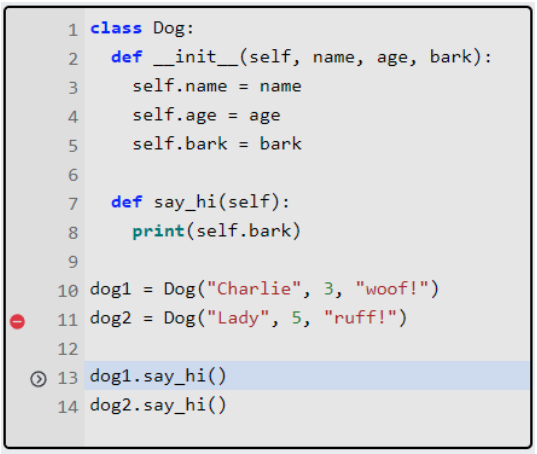
Figur 5.2: Användargränssnitt för menyfältet. Alla knappar till vänster styr exekveringen och knappen till höger genererar en rullgardinsmeny med kodexempel.

5.2.2 Kodrutan

Under menyn finns kodrutan (figur 5.3 och 5.4). Här kan användaren skriva Pythonkod. Om användaren stegar igenom programmet, kommer kodrutan att låsas. Användaren kan alltså inte modifiera koden under stegningen. Raden som är på väg att exekveras är markerad med en grå-blå färg och en pil visas bredvid. Det går även att placera ut brytpunkter till vänster om varje rad, som visualiseras av en röd prick. Om användaren exekverar programmet utan att stega kommer programmet att stanna vid brytpunkten, givet att en brytpunkt har placerats. Det går sedan att fortsätta, antingen genom att stega eller exekvera igenom hela programmet med hjälp av knapparna i menyfältet.



Figur 5.3: Användargränssnitt för kodrutan. Här skriver användaren in sin Pythonkod som sedan går att exekvera. Den röda stopp-ikonen till vänster om rad elva representerar en brytpunkt.



Figur 5.4: Kodrutan med en mörkare nyans för att symbolisera att den är låst. Den aktiva raden under stegningen kan identifieras av både raden som är markerad blå och pil-ikonen till vänster om radsiffran 13.

5.2.3 Terminalen

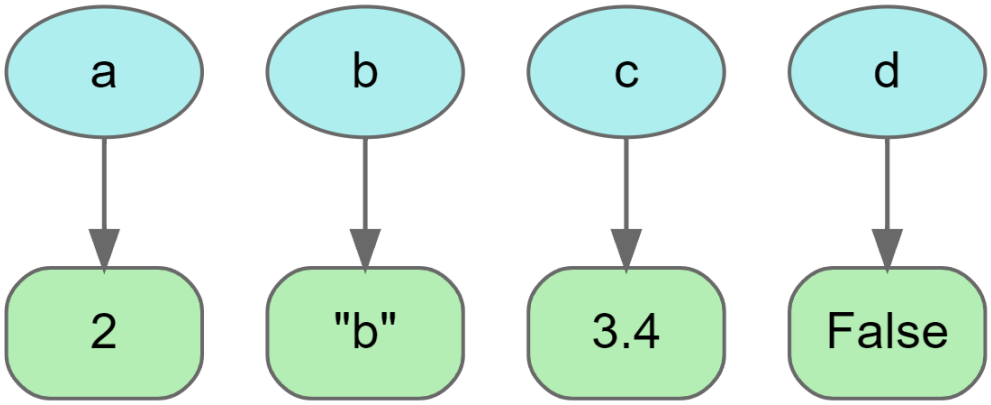
Terminalen visar alla utskrifter som är gjorda med Pythons funktion *print*, samt felmeddelanden om de uppstår. Terminalen är inte en terminal där det går att skriva in kommandon, utan är strikt en utgångskanal. När ny utdata skrivs till terminalen uppmärksammas användaren om detta genom att terminalen får en ljusblå skuggning under ett par sekunder.

5.2.4 Visualiseringsrutan

Visualiseringsrutan, som är placerad på den högra delen av webbsidan, visar en graf över hur variabler och objekt relaterar till varandra. Den visuella komponenten placerades vid sidan av koden för att visa korrelationen mellan dessa.

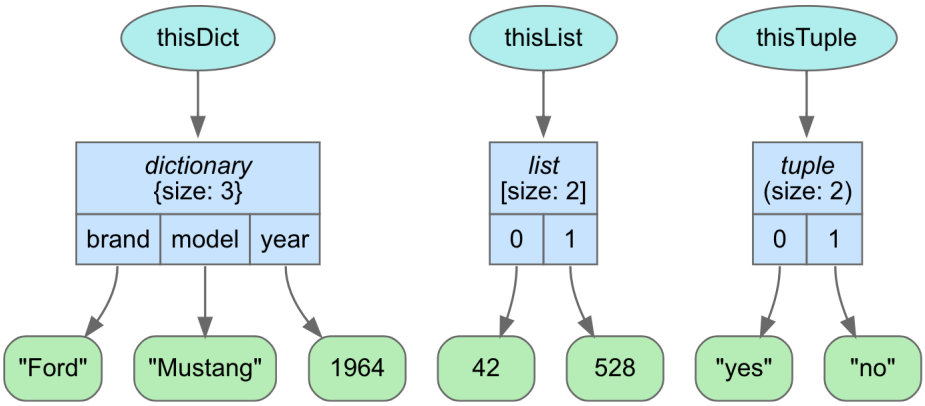
Objekt av datatypen heltal, strängar, flyttal och booleska värden visualiseras på samma sätt. Objekten representeras av en grön cell med rundade kanter, med dess

värde placerat inuti. Alla variabelnamn visualiseras som en turkos oval med namnet inuti. Se exempel i figur 5.5.

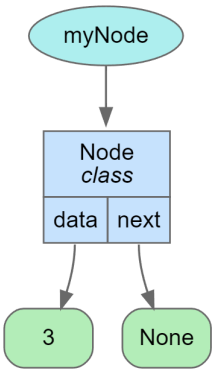


Figur 5.5: Exempel på visualisering av heltal, strängar, flyttal och booleska värden. *a* pekar på heltalsvärdet *2*. *b* pekar på strängen *"b"*. *c* pekar på flyttalet *3.4*, och d pekar på det booleska värdet *False*.

Datatyperna list, tuple, dictionary och klasser visualiseras som en blå cell, med mindre blåa celler inuti. Den yttre cellen anger vilken datatyp objektet är, och av vilken storlek objektet är. De inre cellerna representerar de olika attribut objektet har, som i sin tur pekar på andra objekt. Dessa attribut skiljer sig beroende på vilken datatyp det är. Hos listor och tupler är attributen nummer som anger vilket index objekten har i listan eller tuplen. Hos dictionaries är attributen nycklarna i dictionarien som i sin tur pekar på vilket objekt som är kopplat till den nyckeln. Visualiseringens attribut hos klasser är klassernas programmerade attribut i Python. En skillnad mellan klasser och de andra är dock att hos klasser står klassens namn ovanför datatypen i den yttre cellen. Se exemplet i figur 5.6 för listor, tupler och dictionaries. Se exemplet i figur 5.7 för klasser.



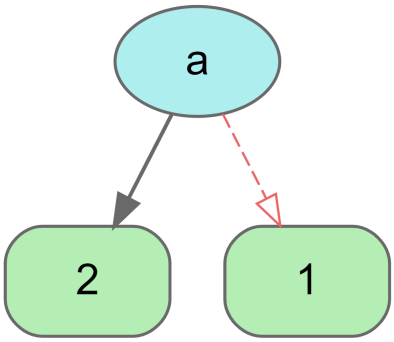
Figur 5.6: Exempel på visualisering av list, dictionary och tuple i LearnyPy. Variabeln *thisdict* pekar på en dictionary som innehåller nycklarna *brand*, *model* och *year*. Dessa pekar i sin tur på *"Ford"*, respektive *"Mustang"* och *1964*. Variabeln *thislist* pekar på en lista som består av heltalen *42* i index 0 och *528* i index 1. Slutligen pekar variabeln *thisTuple* på en tuple som innehåller strängen *"yes"* i index 0 och *"no"* i index 1.



Figur 5.7: Exempel på hur en klass visualiseras i LearnyPy. Här går det att se att variabeln *myNode* pekar på en instans av klassen *Node* som har två attribut, *data* och *next*. *data* pekar i sin tur på *3* medan *next* pekar på ingenting (*None*).

En beskrivande text av grafen läses upp om användaren klickar på visualiseringsrutan och har aktiverad skärmläsare. Grafen som visas ovan hade exempelvis gett uppläsningen: ”Variable ”myNode” points to a class named ”Node” with 2 attributes. Attribute data of ”fixa” points to the integer value 3. Attribute next of ”fixa” points to the NoneType value None.”

I visualiseringen går det även att se ifall en referens byts ut. Detta visualiseras genom att den gamla referenser blir en röd streckad pil. Detta är endast relevant om användaren stegar igenom koden. Se figur 5.8.



Figur 5.8: Exempel på hur det ser ut när en variabel som pekar på ett oföränderligt objekt ändrar värde. Variabeln *a* pekade först på heltalsvärdet *1*, och pekar efter stegning på heltalsvärdet *2*. En ändring av variabeln *a* påverkar alltså inte det oföränderliga objekt som har värdet *1*, utan pekar istället på ett annat objekt med det värdet.

Stegning igenom användardefinierade funktioner resulterar inte i visualisering av de lokala variabler definierade i funktionens räckvidd. Implementering av denna funktionalitet planerades initialt, men blev aldrig genomförd. Mer om detta kan läsas i avsnitt 6.3.

5.3 Informationssidan

Längst upp till höger i LearnyPys gränssnitt i figur 5.1 syns en knapp i form av informations-ikon. Genom att klicka på den knappen navigeras webbplatsen från startsida, vilket presenterades i avsnitt 5.2, till informationssida. Informationssidan presenterar information om LearnyPy såsom syftet av LearnyPy, projektets medarbetare, kända begränsningar och tack till personer som har varit hjälpsamma för detta projekt. Längst ner på informationssidan finns dessutom en länk till webbplatsens tillgänglighetsredogörelse.

5.4 Utvärdering av LearnyPy genom användartestning

Andra omgången av användartester genomfördes i två delar: en bevakad användartestning liknande den första användartestningen och en obevakad som bedrivs i form av en onlineformulär. De bevakade användartesterna genomfördes med samma deltagare som deltog i den första omgången av användartester. Formulären för det obevakade användartestet skickades till studenterna i en introduktionskurs i programmering från Chalmers. Deltagarna i båda omgångarna av användartester tillhörde alltså den avsedda målgruppen för LearnyPy. Sammanfattningar av dessa användartester finns i Bilaga D respektive E.

Sammanfattningsvis så visar testerna på att LearnyPy var uppskattat och många av användarna tyckte generellt att produkten var lättanvänd och lärorik. Detta tyder på att LearnyPy har uppfyllt sitt mål om att bidra som hjälpmedel för studenter att förstå konceptet referenser. Det uppmärksammades dock ytterligare förbättringsområden i detta användartest, och nedan sammanfattas de åtgärder som vidtogs efter andra omgången av användartester:

- Flera användare förväxlade *stopp*-knappen och knappen för att rensa alla brytpunkter. För att förtydliga den sistnämnda knappens funktionalitet så inaktiveras den då det inte finns några brytpunkter. Den aktiveras endast då brytpunkter existerar. Att knappen är inaktiverad visas genom att den görs mindre synlig.
- En användare kommenterade på en bugg: skrollningslisten visas i terminalen även om den är tom. Detta åtgärdades.
- En användare upplevde att det var svårt att se vilket kodexempel som har öppnats. Därför gjordes så att kodexemplets namn står i en kommentar på översta raden i varje kodexempel.

Dessvärre säkerhetsställdes aldrig att resultatet av dessa åtgärder gav positiv respons, då det inte fanns tid till detta under projektets gång. Ingen av åtgärderna anses dock utmana frågan huruvida LearnyPy uppfyller sitt huvudsakliga syfte.

5.5 Filhierarki

Nedan presenteras uppbyggnaden av källkodens filstruktur. I allmänhet så har källkodens filstruktur följt en konvention som finns för React, nämligen att alla filer som är bundna till en viss funktionalitet blir placerade i en separat mapp [52].

Filhierarkin för källkoden kan ses i figur 5.9. Från topp till botten så innehåller *.github/workflows*-mappen de skript som håller logiken för kontinuerlig integrering i GitHub. Vidare så innehåller *public* statiska filer som inte bearbetas när webbapplikationen byggs, som exempelvis de filer som krävs för att exekvera Skulpt i webbläsaren. Den sista mappen i rotkatalogen är *src* och kan betraktas som kärnan för React-applikationen då den innehåller projektets källkod. Inuti *src* finns *assets*-mappen som består av bilder som används av användargränssnittet. *components* innehåller en samling av återanvändbara komponenter för användargränssnittet, såsom knappar och sidhuvud. Dessa komponenter kan användas i flera olika filer i projektet. Därefter kommer *features*-mappen som i sin tur innehåller en del mappar. Varje undermapp i *features* representerar, som namnet antyder, en specifik funktionalitet i webbapplikationen och omfattar alla filer som berör just den funktionaliteten, exempelvis CSS, JavaScript och test-filer. Till skillnad från *components* så är inte målet med komponenterna i *features* att vara återanvändbara. *skulpt-wrapper* är en ytterligare mapp i *src* som behandlar de filer som hanterar kommunikationen med Skulpt. Dessa filer tillhandahåller den information som användargränssnittet behöver för att illustrera grafen och utdatan från den kod som användaren har matat in. Slutligen har *src* även en *pages*-mapp. Varje fil i den här mappen motsvarar en sida som kan besökas i webbapplikationen.

Figur 5.9: Filhierarkin för källkoden.

6
Diskussion

I detta kapitlet diskuteras problem som har uppstått under projektets gång och hur vi har löst dem. Vi lyfter även etiska problem med LearnyPy. Det diskuteras också om olika val vi har behövt göra, och hur dessa val har påverkat projektet och LearnyPy.

6.1 Samhällsdiskussion

Resultatet av projektet har en stor potentiell samhällsnytta inom lärandet av programmering. I den digitala etablering som råder är det en fördel för vidare utveckling desto fler som lär sig programmering. LearnyPy kan underlätta för lärare i och med att de får ytterligare ett verktyg till sitt förfogande, vilket är nyttigt då bakgrunden belyser att människor lär sig olika bra av olika undervisningsstrategier. Sannolikt är även att inlärningstakten generellt kan öka med hjälp av LearnyPy, vilket är samhällsnyttigt då det frigör mer disponibel tid för nya kunskaper hos studenterna.

Ett motargument mot att digitalisera olika processer är att det utplånar yrkesroller [53]. Det skulle kunna ses som ett etiskt dilemma för detta projekt, då resultatet följer trenden att frånta en yrkesroll dess tidigare uppgift (att enbart läraren förklarar hur relationer mellan objekt och variabler fungerar). Men projektet avser istället att underlätta för läraren att bedriva en mer mångsidig undervisning.

6.2 Visualisering av koden

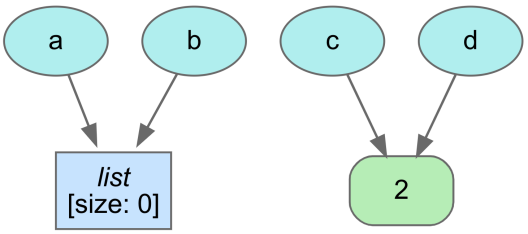
Detta avsnitt diskuterar problem och beslut som uppstod i samband med visualisering av koden. Vi startar med en diskussion kring hur vi skulle visualisera oföränderliga objekt. Sedan presenteras hur vi valde att justera minneshanteringen för heltal och flyttal, för att fortsätta vidare med att diskutera ett problem med att hålla visualiseringen uppdaterad med vad som står i kodrutan. Till sist diskuteras hur vi visualiserar dictionaries.

6.2.1 Oföränderliga objekt

Under projektets gång har det varit en diskussion kring hur vi ska visualisera skillnaden mellan föränderliga och oföränderliga objekt. I början visualiserades oföränderliga objekt genom att det endast fanns ett unikt objekt för varje oföränderligt objekt, likt hur Python förvarar dem i minnet. Om det fanns en variabel *a* som pekade på *2* och en variabel *b* som pekade på *2*, så fanns det endast en nod innehållande *2* som båda variablerna pekade på.

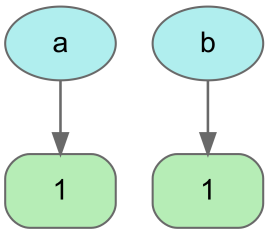
Problemet med denna visualiseringen var att det inte är intuitivt vad skillnaden

mellan föränderliga och oföränderliga objekt är. Se exempelvis figur 6.1 där det kan tänkas möjligt att användaren tolkar det som att a och b kan förändra listan och att d och c kan förändra heltalet, eftersom variabel-paren visualiseras på samma sätt.



Figur 6.1: Visualisering av referenser till ett föränderligt objekt (en lista), och ett oföränderligt objekt (ett heltal).

Ett förslag för att undvika denna problematik var att istället skapa nya noder av alla oföränderliga objekt varje gång de tilldelas en variabel, för att visa att inga variabler kan påverka varandras referens via detta objekt, se figur 6.2. Däremot finns nackdelen av detta att det inte speglar en korrekt referenshantering enligt Pythons implementation, eftersom oföränderliga objekt kan ha flera variabler som pekar på just dem.



Figur 6.2: En alternativ visualisering där oföränderliga objekt dupliceras, för att tydliggöra att a och b inte kan påverka varandras värde.

Det diskuterades om ett alternativ där båda visualiseringarna skulle implementeras, och användaren fick välja vilken visualisering hen ville se. Det var dock inte självklart vilket av alternativen som skulle vara standard. Vi ville även ha LearnyPy så simpel att använda som möjligt, och att ha två alternativa visualiseringar hade krävt förklarande text för vad skillnaderna var, vilket vi ville undvika.

I samråd med handledare landade vi till slut i att enbart visualisera alternativet med korrekt referenshantering enligt Python, som visas i figur 6.1, då detta är viktigt för att uppfylla projektets syfte. Problemet kring att användaren kan bli förvirrad över vilka variabler som kan påverka vilka objekt är något som ligger i att förstå vilka datatyper som är föränderliga och inte. Om användaren vill se om oföränderliga påverkar varandra kan hen exekvera koden. Vi funderade på att visa skillnaden mellan oföränderliga och föränderliga objekt genom att märka deras noder i grafen med någon symbol, men detta infördes inte då det upplevdes som att det inte finns någon intuitiv symbol för detta. Att införa egna märkningar kan ge sidoeffekten att gränssnittet blir för komplext för en nybörjarprogrammerare att sätta sig in i.

För att förtydliga skillnaden på när ett objekt har modifierats eller bytts ut, men

ändå behålla en korrekt representation av referenser, implementerades en visualisering av något vi valde att kalla ”döda referenser”. Dessa döda referenser skapas endast då användaren stegar igenom koden rad för rad och en ny tilldelning sker. Den döda referensen illustrerar vad som refererades till innan den nya tilldelningen skedde. Ta som exempel följande kodstycke:

```
a = 1
a += 1
```

Här finns risk att användaren tror att objektet som *a* refererar till kommer modifieras i och med ökningen. Den döda referensen kan i detta fall förtydliga att *a* inte längre refererar till sin ursprungliga tilldelning utan till ett objekt som innehåller resultatet av ökningen. En visualisering av döda referenser kan ses i figur 5.8 i avsnitt 5.2.4.

6.2.2 Justering av minneshantering för heltal och flyttal

Vår lösning för att hantera och spara alla variabler, objekt och dess referenser, bygger på att vi jämför de JavaScript-objekt som Skulpt har skapat som representation av Pythons objekt. Om exempelvis två olika listor har initialiserats, kommer en jämförelse mellan dessa visa att de inte är samma objekt. Om däremot flera variabler har tilldelats samma objekt innehållande en lista, kommer en jämförelse mellan dessa variablers JavaScript-objekt visa att det är samma objekt som variablerna refererar till. Denna lösning fungerar mycket bra vid jämförelse av exempelvis föränderliga objekt, strängar och booleska värden. Däremot uppstod problem vid hantering av heltal och flyttal. Ta som exempel följande kod:

```
x = 257
y = 257+1-1
print(x is y)
```

Då vi testar att kompilera och exekvera denna kod med Python visar utskriften att x och y refererar till samma objekt. Detta tyder på att Pythons kompilator optimerar hanteringen av objekt så att det inte sker duplicering av heltalsobjekt innehållande samma värde. Det förväntade resultatet i grafen är därför att både *x* och *y* ska ha en kant till samma objekt. I vår initiala lösning skapades istället två olika JavaScript-objekt som båda innehöll heltalet *257*, då det var dessa objekt Skulpt genererade. En jämförelse av dessa två objekt visade alltså att de ej var lika. Samma problem med duplicering av flyttal uppstod med vår initiala lösning. Även detta upplevdes inkorrekt då liknande test som det ovan visade att hantering av flyttal optimeras så att de ej dupliceras. För att undvika detta förvirrande resultat implementerades istället en lösning som justerar så att duplicering av objekt för heltal och flyttal aldrig sker. Detta ger ett förutsägbart mönster för vilka objekt som skapas och efterliknar mer de optimeringar vi tror Pythons kompilator gör.

6.2.3 Rendering av visualiseringen

LearnyPy låter användaren aktivera uppdatering av visualiseringen genom att exekvera, eller stega igenom, koden. Till en början valde vi att följa trenden hos andra etablerade utvecklingsmiljöer, genom att användaren kunde förändra koden efter att

ha påbörjat avbuggning av koden, för att undvika att lära användaren fel programmeringspraxis. Dock visade användartesterna på att detta inte var lämpat för vår målgrupp, då de trodde att koden i jämsides visualiseringen alltid var den som representerades. Vi diskuterade kring om vi ändå skulle behålla detta, eftersom detta förmodligen enbart är ett problem för att användarna har mycket liten programmeringsvana. Vi landade dock i att det trots allt är de som är vår målgrupp, och vi vill göra det så enkelt som möjligt för dem att använda LearnyPy genom att undvika att försöka lära dem allt på en gång. Ett alternativ var då att låsa koden från att kunna modifieras under exekvering. I vårt fall måste användaren klicka på ”stopp”-knappen för att kunna redigera koden när en exekvering är pågående. På så vis finns aldrig någon visualisering jämsides med kod som inte är den kompilerade och visualiserade koden. Användartesterna som genomfördes med detta implementerat visade goda resultat.

6.2.4 Datatypen dictionaries

Pythons datatyp dictionary innehåller par av en nyckel och ett värde [54]. I detta par är både nyckeln och värdet en referens till ett objekt. För att visa detta sanningsenligt i vår graf skulle både nyckeln och värdet ha en kant till det objekt som refereras till. Kanter för nycklar gör så att grafen upplevs mer svårläslig och därför valdes en alternativ lösning där nyckeln istället representeras direkt i noden. Utseendet blir då mer likt exempelvis listor, där var index representeras av en siffra direkt i noden. En nackdel med detta är att vi inte längre håller oss till en helt korrekt representation av referenser. Vi argumenterar dock för att vår lösning har en större fördel med sin enkla visualisering, än denna nackdel. Eftersom alla objekt som väljs till nycklar måste vara oföränderliga [54] kan de inte vara föremål för sidoeffekter. Alltså, trots att exempelvis en variabel kan referera till samma objekt som används som nyckel, kan inte en modifiering av denna variabel leda till att nyckelns objekt förändras. Därför blir relationen mellan dessa mindre viktiga att förmedla.

6.3 Val av Skulptgren

Den version av Skulpt som används i LearnyPy är inte samma version som den senaste på Skulpts GitHub-sida. Detta beror på att vi vill kunna visualisera lokala variabler, vilket det inte fanns något stöd för i Skulpts huvudgren vid projektets utförande. Det fanns dock en annan gren som implementerade ett sätt att hämta lokala variabler, vilket gjorde att vi bytte till denna gren. Detta gjorde vi genom att vi laddade ned den lokalt för att även kunna göra korrigeringar i den. Dock innebar detta några problem, varav de flesta kommer ifrån att grenen inte är uppdaterad sedan år 2020. Det största problemet är att vår applikation inte automatiskt kommer att uppdatera i synk med uppdateringar till Skulpt. Om vi istället använde huvudgrenen hade vi kunnat göra så att LearnyPy hämtade Skulpt-filerna direkt från Skulpts webbplats.

Även vissa buggar som kan vara åtgärdade i Skulpt är inte nödvändigtvis lösta i denna grenen, då den är väldigt gammal. Ett exempel på detta var funktionen

round(). Detta är en funktion vars syfte är att avrunda ett flyttal till ett heltal, eller till ett annat flyttal med en angiven precision. I denna grenen returnerade funktionen *round()* alltid ett flyttal, vilket inte är i enighet med Pythons implementation av *round()* [55]. Detta löstes dock genom att korrigera funktionen i Skulpts fil *float.js*, men det kan finnas andra problem vi inte är medvetna om i dagsläget.

En fördel med att använda denna lokala gren är dock att webbplatsen inte löper någon risk att sluta fungera efter en uppdatering. I en situation där Skulpt skulle publicera en uppdatering som omstrukturerar kodbasen så att vår kod inte fungerar längre, skulle detta inte påverka oss då vi använder vår egna version lokalt. Detta innebär att vi inte kommer att ha någon nertid samtidigt som problemet åtgärdas.

Bytet till denna gren resulterade alltså i att vi kunde hämta ut information gällande variabler och objekt för den lokala räckvidden. Ett förslag för visualisering av detta var att införa en symbol för globala och lokala variabler i noderna. Dock kunde vi inte hitta symboler som intuitivt skulle kunna representera detta. Ett annat förslag var att dela upp visualiseringsrutan i två delar, en för vardera globala och lokala variabler. Problematiken med detta förslag låg dels i att arrangera delarna så att visualiseringen fortfarande var tydlig. Frågetecken kring hur hanteringen av objekt som refererades till från både globala och lokala variabler fanns också. Slutligen beslutades att helt enkelt visualisera lokala variabler på samma sätt som globala. Dock hann vi inte med att implementera denna lösning innan projektets slut. Vid projektets slut finns en fungerande logik för att hämta data gällande en lokal räckvidd, men inte en fungerande visualisering.

6.4 Användningen av Skulpts avbuggare

Avbuggaren som används i LearnyPy bygger på både det inbyggda suspensionssystemet, som hanterar avbrott under avbuggningen, och den externa filen *debugger.js* i Skulpt. Implementeringen av avbuggaren har varit komplicerad, och en del problem har dykt upp under projektets gång.

Det var komplicerat att få stegningen att fungera tillsammans med funktionen att exekvera hela programmet. Exempelvis, om användaren stegade igenom programmet var hen sedan tvungen att klicka på ”kör”-knappen två gånger för att kunna exekvera igenom hela programmet igen. Detta var på grund av att programmet tekniskt sett redan exekverade när användaren stegat igenom, och det löste sig med en extra check för att se om programmet redan var klart.

Ett annat problem var användardefinierade funktioner. I basversionen av *debugger.js* går det inte att stega igenom användardefinierade funktioner, utan att få hela programmet att avsluta efter den användardefinierade funktionen. Problemet verkar vara att suspensionstacken som är definierad i *debugger.js* inte beter sig på rätt sätt. Suspensionstackens roll är att hålla reda på vilka rader kod som ska exekveras i vilken ordning. Det översta elementet i stacken tas bort och exekveras, samtidigt som nästa rad läggs till i stacken i väntan på att bli exekverad. När programmet går in i

en funktion skapas en ny suspension vid linjen där programmet gick in i funktionen, och en till underordnad suspension för funktionen. Denna underordnade funktionen kommer att läggas till överst på stacken så att den exekveras först, och även som ett attribut till den övre suspensionen. Detta för att programmet ska veta vilken suspension som utlöste den nya suspensionen, så att programmet kan återvända till den ursprungliga raden kod. När funktionen är klar så försvinner den underordnade suspensionen från stacken helt och hållet, och programmet fortsätter att exekvera original suspensionen.

Det som hände i verkligheten var dock inte detta. Istället för att Skulpt skapade en attribut till den överordnade suspensionen med den underordnade, ersatte Skulpt istället den överordnade suspensionen med den underordnade. Detta resulterade i att Skulpt tappade bort informationen om vart programmet gick in i funktionen, och fick Skulpt att tro att programexekveringen var klar efter funktionen. Detta eftersom att suspensionstacken var tom, vilket får programmet att avslutas. Genom att korrigera detta i *debugger.js*, löste sig problemet.

Lösningen av det tidigare problemet ledde dock till upptäckten av ett ytterligare problem. I de situationer när en funktions returneringsvärde tilldelas till en variabel så visade det sig att visualiseringen för variabler, objekt och relationer mellan dem inte återspeglade programmets nya tillstånd. Efter en del felsökning så upptäcktes det att tillståndet av programmet som har matats in, som är förvarad i det globala objektet *Sk* i JavaScript, inte uppdateras i tid. Detta är problematiskt då LearnyPy är beroende av att *Sk* är uppdaterad efter varje stegning eftersom detta objekt avgör vad som visualiseras. Problemet berodde på en funktion i *debugger.js* som ansvarar för stegningen. Denna funktion innehöll nämligen en asynkron metod som orsakade det oönskade icke-sekventiella exekveringsflödet. Detta löstes dock genom att den asynkrona metoden inväntades manuellt för att säkerställa ett sekventiellt flöde, som i sin tur medförde att *Sk* faktiskt avspeglar programmets nya tillstånd.

Det är svårt att tro att de ovannämnda problemen var ett misstag på Skulpts sida, då Skulpt har en rigorös process för att godkänna kod [23]. Dock har modifieringarna av koden i *debugger.js* inte haft några negativa effekter på kompilatorn och programmet, och vi anser därför att de modifieringar som har gjorts inte är några problem.

6.5 Alternativ till Skulpt

I avsnitt 6.4 diskuterades de problem som har uppstått med Skulpt och det är tydligt att användningen av detta bibliotek inte har varit felfritt. Det fanns många anledningar till varför just Skulpt valdes i början av projektet, som beskrivits i avsnitt 4.1.1. Ett argument var att vi ville undvika att skapa en kompilator från grunden upp eftersom det inte var huvudfokuset av projeketet. Däremot ligger nog det främsta skälet i att vi blev presenterade till biblioteket Skulpt vid ett tidigt skede i projektet. Nu, i efterhand, så borde vi ha kartlagt alternativen som finns för att kompilera och exekvera Pythonkod i webbläsaren, och därefter genomfört en analys

för att identifiera det mest lovande biblioteket.

Vid en senare fas i projektet så upptäcktes det ett annat bibliotek, vid namn Pyodide [40], som likt Skulpt kompilerar och exekverar Pythonkod direkt i webbläsaren. Däremot verkar Pyodide vara bättre än Skulpt i flera aspekter. Till exempel så ger dokumentationen för Skulpt inte en heltäckande beskrivning av hur källkoden fungerar, hur den är uppbyggd och hur man använder den. Detta medförde att Skulpts källkod fick undersökas manuellt, för att få en uppfattning över systemets uppbyggnad, och hur viktig information skulle fås ut. Detta var en tidskrävande process. Pyodide har dock en omfattande och detaljerad dokumentation. Exempelvis så är det tydligt hur variablerna definierade i Python kan nås från JavaScript, vilket inte var fallet för Skulpt.

Ett annat problem som uppstår med Skulpt är om LearnyPy i framtiden ska kunna behandla mer komplicerad kod genom användningen av externa Python-paket. Vid genomförandet av projektet så var Skulpt väldigt begränsad gällande externa paket, vilket beror till stor del på att Skulpt är en JavaScript-implementation av Python. Detta resulterar i att varje paket som önskas användas i Skulpt måste översättas till JavaScript så att det är kompatibelt med Skulpts miljö. Pyodide drabbas däremot inte av de problem som Skulpt har med externa paket. Detta beror på att Pyodide använder sig av CPython som har kompilerats till WebAssembly [40]. Genom Pyodides tillvägagångssätt så möjliggör det användningen av alla externa Python-paket, givet att de är publicerade på Python Package Index (PyPi) och att det finns Python-hjul (eng. Python Wheel) tillgängliga [40].

Med projektets syfte och avgränsningar i åtanke så kan Skulpt ändå anses vara tillräckligt. Initialt så var visionen att LearnyPy endast behandlar några rader Pythonkod, vilket Skulpt är fullt kapabel till. Däremot kan ett potentiellt vidare arbete vara att ersätta Skulpt med Pyodide för att säkerställa att LearnyPys framtida funktionaliteter inte blir begränsade av Skulpt.

6.6 Uppbyggnad av testerna

Testerna i LearnyPy är uppdelade i flera testfiler. Varje testfil avser varsitt ansvarsområde och innehåller enbart tester som är relevanta för sitt ansvarsområde. Exempelvis, i *breakpoints.test.js* testades enbart interaktioner med brytpunkterna, som filnamnet tyder på. Det finns däremot en del problem med implementeringen av testerna trots att testningen är fungerande.

Som det har förklarats i avsnitt 4.2.4, så kräver alla tester ett existerande användargränssnitt för att exekveras. Som ett resultat skrivs testerna på sådant sätt att vi först interagerar med gränssnittet programmatiskt för att få ut det önskade tillståndet. Därefter testas resultatet från gränssnitt eller så anropas en funktion, beroende på vad som ska testas. Kravet på interagering med användargränssnittet är dock problematiskt då även funktionalitet som är oberoende av användargränssnittet måste testas med ett gränssnitt. Testningsstrukturen skulle alltså kunna förbättras avse-

värt om det hade varit möjligt att importera Skulpt som en oberoende JavaScript-modul.

6.7 Kodkvaliteten i LearnyPy

Under projektets gång så har vi eftersträvat att upprätthålla kvaliteten i källkoden. Detta gjordes bland annat för att säkerställa att koden är lätt att underhålla samt att det ska vara möjligt att lägga till ny funktionalitet utan att tidigare kod behöver modifieras. Ett tydligt exempel på hur vi har uppnått detta är att all kod som berör kompilatorn är separerad och oberoende av användargränssnittet. Det är alltså möjligt att byta ut React till något annat dylikt bibliotek om det finns behov i framtiden. En annan sak som vi har haft i åtanke är hur nya utvecklare ska kunna arbeta vidare på LearnyPy. Några åtgärder som gjordes för att främja vidareutvecklingen var att filhierarkin för källkoden följer Reacts konventioner, och vi eftersträvade att dokumentera mycket samt att följa en enhetlig stil. På detta vis minskar förhoppningsvis tröskeln, för en utomstående, att förstå hur källkoden fungerar för att sedan kunna vidareutveckla LearnyPy.

6.8 Genomförandet av tillgänglighetsanalysen

En tillgänglighetsanalys av en webbplats kan genomföras på två olika sätt: antingen genom en intern självbedömning eller med hjälp av en extern aktör som exempelvis Funka [28]. Det senare alternativet bör prioriteras eftersom det blir en mer rigorös och objektiv utredning. Däremot valdes det tidigare alternativet, vilket beror på att projektets icke-existerande budget inte kan täcka kostnaderna för en tillgänglighetsanalys genomförd av en extern aktör. Detta innebär att LearnyPy endast är testat på våra privata datorer, vilket medför att vi kan ha missat problem som visar sig i miljöer vi inte testade i. På grund av att analysen är genomförd av oss, som dessutom inte har genomfört någon kurs inom tillgänglighet gällande användargränssnitt, så kan inte heller fullständigheten eller korrektheten garanteras.

Ett exempel på en brist är att vi inte kan säkerställa att vi har bearbetat alla riktlinjer som krävs för att uppnå nivå AA i WCAG 2.1. Detta beror på att vi under en tidig fas av projektet gjorde ett utdrag av kriterierna från WCAG 2.1 som kändes relevanta för LearnyPy. Dessa kriterier lade sedan grunden för genomförelsen av tillgänglighetsanalysen då det var enbart dessa kriterier som granskades. Problemet som kan ha uppstått är att vissa riktlinjer som inte uppfattades vara betydelsefulla för LearnyPy i början av projektet kan senare ha blivit aktuella då LearnyPy förändras ständigt med ny funktionalitet.

Det är även värt att nämna att det inte har varit enkelt med granskningen av de utvalda kriterierna. Vid flera tillfällen upplevdes kriterierna vaga vilket försvårar bedömningen av om ett kriterium har uppnåtts eller inte. Detta verkar dock vara ett känt problem sedan tidigare eftersom WCAG har fått kritik om oklarheten och svårigheten att använda dessa riktlinjer för att förbättra tillgängligheten av en

webbplats [56], [57]. Det blir alltså en tolkningsfråga om en riktlinje är uppfylld eller ej, vilket kan ha en negativ inverkan på analysens korrekthet. Den slutliga tillgänglighetsredogörelsen presenteras i bilaga F.

6.9 Språkval

LearnyPy är utformad på engelska, då programmering oftast associeras med engelska. Engelska öppnar också upp LearnyPy för en större målgrupp, i jämförelse om vi hade utformat den på svenska vilket hade varit det andra alternativet. Oavsett vilket språk vi hade valt, så innebär det dessvärre att de användare som inte behärskar det implementerades språket kan tänkas bli exkluderade. Därav är att låta användaren välja språk själv en lösning på problemet som kan tänkas vara ett vidare arbete. Mer förslag på förbättringsområden diskuteras i nästa kapitel.

7
Vidare arbete

I detta kapitel introduceras några utvecklingsmöjligheter hos slutprodukten. Kapitlet startar med att föreslå en kombinationsprodukt av LearnyPy och en annan webbplats som delar ambitionen att hjälpa nybörjarprogrammerare. Sedan lyfts ett möjligt tillägg till slutprodukten gällande den stegvisa exekveringen, med ambitionen att tillämpa fler standardfunktionaliteter hos avbuggaren. Kapitlet avslutas med att uppmuntra ett framtida vidare arbete kring användarvänligheten. 6

7.1 Kombination med CodingBat

CodingBat [58] är ytterligare en webbplats som erbjuder användaren att programmera i webbläsaren, men innehåller ingen visualisering av koden. Däremot har CodingBat en del genomgångar av grundläggande programmeringsområden, i form av texter och videos. Då CodingBats webbplats även erbjuder mängder av övningsuppgifter för Python, med möjlighet att få ledtrådar och rätta lösningarna till dessa, är det ett perfekt forum för nybörjare att träna programmering på. Om visualisering av relationer mellan objekt och variabler hade funnits till förfogande för användaren hade det förmodligen kunnat hjälpa nybörjaren att lösa uppgifter, men också att grundligt förstå sin egen lösning. Detta gäller framförallt de uppgifter relaterade till föränderliga objekt och alias-problematik. Ett samarbete mellan LearnyPy och CodingBat hade kunnat resultera i programmering i webbläsaren, övningsuppgifter, genomgångar, rättning och visualisering av minneshanteringen. Denna kombination hade troligtvis resulterat i en ännu bättre produkt än vad webbplatserna i nuläget erbjuder enskilt för att uppfylla det gemensamma syftet; att hjälpa användare att lära sig programmering.

7.2 Utöka avbuggaren

En möjlig utökning hos projektets avbuggare är att möjliggöra att kunna starta programmet med en viss hastighet. Användaren ska kunna ange hur många steg i sekunden användaren vill att programmet ska exekvera i, så kan programmet stega framåt automatiskt utan att användaren behöver klicka på ”steg”-knappen. Detta ger dock inte något nytt i programmet, eftersom användaren kan stega manuellt, men det kan förbättra användarupplevelsen. Dessutom finns det möjligheter till att införa en ”stega-tillbaka”-knapp, som möjliggör att stega baklänges i programmet. Vi implementerade aldrig detta eftersom vi ville efterlikna hur en avbuggare vanligtvis ser ut, men en ”stega-tillbaka”-knapp är definitivt något som kan tänkas gynna användaren.

7.3 Användargränssnitt

Ytterligare en aspekt av produkten som hade kunnat gynnas av vidare arbete är användargränssnittet. Detta är inte oväntat då det var en avgränsning för detta projekt att funktionalitet var överordnat arbete med utseendet. Produkten utvärderades med användartest som gav positiva resultat, vilka tyder på att användargränssnittet duger för att förmedla funktionalitet och information. Däremot var det en relativt liten process av användartester, likaså av kartläggande av målgruppens förkunskaper. Inget användartest gjordes med en person som använder någon av de funktionaliteter produkten har för funktionsnedsatta användare. Ett vidare projekt hade kunnat fokusera på och genomföra en mer gedigen process av dessa aspekter för att säkra och förbättra användarvänligheten.

8
Slutsats

Syftet med projektet var att utveckla en produkt som främjar undervisningen av programmering, genom att produkten visualiserar relationer mellan variabler och objekt i Python. Resultatet tyder på att syftet är uppfyllt. Det fanns tre frågeställningar för att vägleda arbetet, och de har uppfyllts till stor utsträckning.

Den första frågeställningen handlade om hur relationer mellan variabler och objekt i Pythonkod kan visualiseras pedagogiskt. Detta är en subjektiv formulering, men användartesternas resultat visar på att LearnyPy är enkel att förstå, vilket kan tolkas som att applikationen är pedagogisk för sitt ändamål. Det är dock möjligt att det finns alternativa visualiseringar som hade upplevts lika eller ännu mer användarvänliga och pedagogiska. Det är ett vidare arbete inom design-fältet att utforska alternativa visualiseringar för att förbättra den pedagogiska aspekten.

Den andra frågeställningen berörde hur applikationen ska göras tillgänglig för allmänheten, för att bemöta den breda målgruppen. Applikationens källkod och plattform är tillgänglig att besöka via internet, och LearnyPy är därmed tillgänglig för allmänheten. Gällande om innehållet av produkten är tillgängligt för allmänheten, spelar anpassningar för funktionsnedsättningar stor roll. Vi har eftersträvat officiella standarder för tillgänglighet gällande funktionsnedsättningar hos webbplatser. Som diskussionskapitlet lyfter, är det dock möjligt att vår process kring funktionsnedsättningar och tillgänglighet inte är komplett och vi reserverar oss för att frågan inte är uppfylld till fullo, även om förhoppningen är så.

Den tredje och sista frågeställningen lyfte hur applikationen ska vara möjlig att vidareutvecklas av andra. LearnyPy är möjlig att utöka för utomstående, eftersom källkoden finns publik. Det strukturerade arbetet med kontinuerlig integrering, filhierarki och tester underlättar också för vidare arbete i kodbasen.

Slutligen hoppas vi att LearnyPy kommer att bidra till att utbilda programmering, samt underlätta och utveckla den ökade programmeringsundervisningen.

Källförteckning

[1] M. Ottoson, ”Därför är det viktigt med programmering i skolan,” *Internetstiftelsen*, [Online]. Okt. 29. 2020. Tillgänglig: https://userway.org/blog/the-future-of-web-content-accessibility-guidelines-wcag30/, (hämtad: 2022-02-25).

[2] ”Beyond Point and Click: The Expanding Demand for Coding Skills,” Burning Glass Technologies, Boston, MA, USA, 2016. [Online]. Tillgänglig: https://www.burning-glass.com/wp-content/uploads/Beyond_Point_Click_final.pdf, Hämtad: 2022-03-03.

[3] E. Lahtinen, K. Ala-Mutka och H. M. Järvinen, ”A study of the difficulties of novice programmers,” *ACM SIGCSE Bulletin*, vol. 37, nr. 3, ss. 14-18, Sep. 2005, doi: 10.1145/1151954.1067453.

[4] A. Cole, ”What Do You Need on Your Computer as a Coding Beginner?,” *Simple Programmer*, [Online]. Jun. 10, 2019. Tillgänglig: https://simpleprogrammer.com/what-you-need-computer-coding-beginner/ (hämtad: 2022-04-17).

[5] L. Ma, J. Ferguson, M. Roper och M. Wood, ”Investigating and improving the models of programming concepts held by novice programmers,” *Computer Science Education*, vol. 21, nr. 1, ss. 57-80, Mar. 2011, doi: 10.1080/08993408.2011.554722.

[6] K. Angelov, personlig kommunikation, Jan. 2021.

[7] A. Salisu och E. N. Ransom, ”The role of modeling towards impacting quality education,” *International Letters of Social and Humanistic Sciences*, vol. 32, ss. 54-61, Jun. 2014, doi: 10.18052/www.scipress.com/ILSHS.32.54.

[8] L. Jones, ”Pointers in Python: What’s the Point?,” *Real Python*, [Online]. Tillgänglig: https://realpython.com/pointers-in-python/ (hämtad: 2022-02-02).

[9] D. A. Brinkerhoff, ”Memory Management: Stack And Heap,” *Weber State University*, [Online]. Tillgänglig: https://icarus.cs.weber.edu/~dab/cs1410/textbook/4.Pointers/memory.html (hämtad: 2022-04-28).

[10] D. A. Brinkerhoff, ”Stacks And Stack Operations,” *Weber State University*, [Online]. Tillgänglig: https://icarus.cs.weber.edu/~dab/cs1410/textbook/7.Arrays/stack.html (hämtad: 2022-03-03).

[11] OpenDSA, ”Heap Memory,” 2021. [Online]. Tillgänglig: https://opendsa-server.cs.vt.edu/OpenDSA/Books/CS2/html/HeapMem.html (hämtad: 2022-03-03).

[12] M. Martin, ”Stack vs Heap: Know the Difference,” *Guru99*, [Online]. Mar. 5, 2022. Tillgänglig: https://www.guru99.com/stack-vs-heap.html (hämtad: 2022-05-11).

[13] R. Johansson, *Grundläggande datorteknik*, 1. uppl., Lund, Sverige: Studentlitteratur, 2016, kap. 8, ss. 197-241.

[14] A. C. Stross-Radschinski, ”python - a programming language that changes the world,” *Python Software Foundation*, [Broschyr], Tillgänglig: https://brochure.getpython.info/media/releases/psf-python-brochure-vol.-i-final-download.pdf/view (hämtad: 2022-05-09).

[15] Python Software Foundation, ”Python Implementations,” 2022. [Online]. Tillgänglig: https://wiki.python.org/moin/PythonImplementations (hämtad: 2022-05-12).

[16] Python Software Foundation, ”Common Object Structures,” 2022. [Online]. Tillgänglig: https://docs.python.org/3/c-api/structures.html (hämtad: 2022-03-22).

[17] Python Software Foundation, ”Memory Management,” 2022. [Online]. Tillgänglig: https://docs.python.org/3/c-api/structures.html (hämtad: 2022-03-03).

[18] P. Conrad, ”Python Function Calls and the Stack,” *University of California*, [Online]. 2010. Tillgänglig: https://sites.cs.ucsb.edu/~pconrad/cs8/topics.beta/theStack/02/ (hämtad: 2022-03-03).

[19] V. Yourdanov, ”Python Basics: Mutable vs Immutable Objects,” *Towards Data Science*, [Online]. Feb. 3, 2019. Tillgänglig: https://towardsdatascience.com/https-towardsdatascience-com-python-basics-mutable-vs-immutable-objects-829a0cb1530a (hämtad: 2022-04-19).

[20] H. J. S. Germain, ”Pointers and References,” *School of Computing - The University of Utah*, [Online]. 2010. Tillgänglig: https://www.cs.utah.edu/~germain/PPS/Topics/pointers_and_references.html (hämtad: 2022-02-25).

[21] J. Elkner, A. B. Downey och C. Meyers, ”Lists,” i *How to Think Like a Computer Scientist: Learning with Python*, 1. uppl., Wellesley, MA, USA: Green Tea Press, 2002, kap. 8, ss. 81-94. Tillgänglig: https://www.greenteapress.com/thinkpython/thinkCSpy.pdf, Hämtad: 2022-02-23.

[22] Skulpt Community, ”skulpt/skulpt,” 2021. [Online]. Tillgänglig: https://github.com/skulpt/skulpt (hämtad: 2022-03-03).

[23] Skulpt Community, ”skulpt/skulpt/CONTRIBUTING.md,” 2019. [Online]. Tillgänglig: https://github.com/skulpt/skulpt/blob/master/CONTRIBUTING.md (hämtad: 2022-05-12).

[24] D. Abramov, ”Running Tests,” *Create React App*, [Online]. Jul. 7, 2021. Tillgänglig: https://create-react-app.dev/docs/running-tests/ (hämtad: 2022-05-11).

[25] Meta Open Source, ”Testing Overview,”. [Online]. Tillgänglig: https://reactjs.org/docs/testing.html (hämtad: 2022-03-24).

[26] Netlify, ”Build the future of the web,”. [Online]. Tillgänglig: https://www.netlify.com/ (hämtad: 2022-03-24).

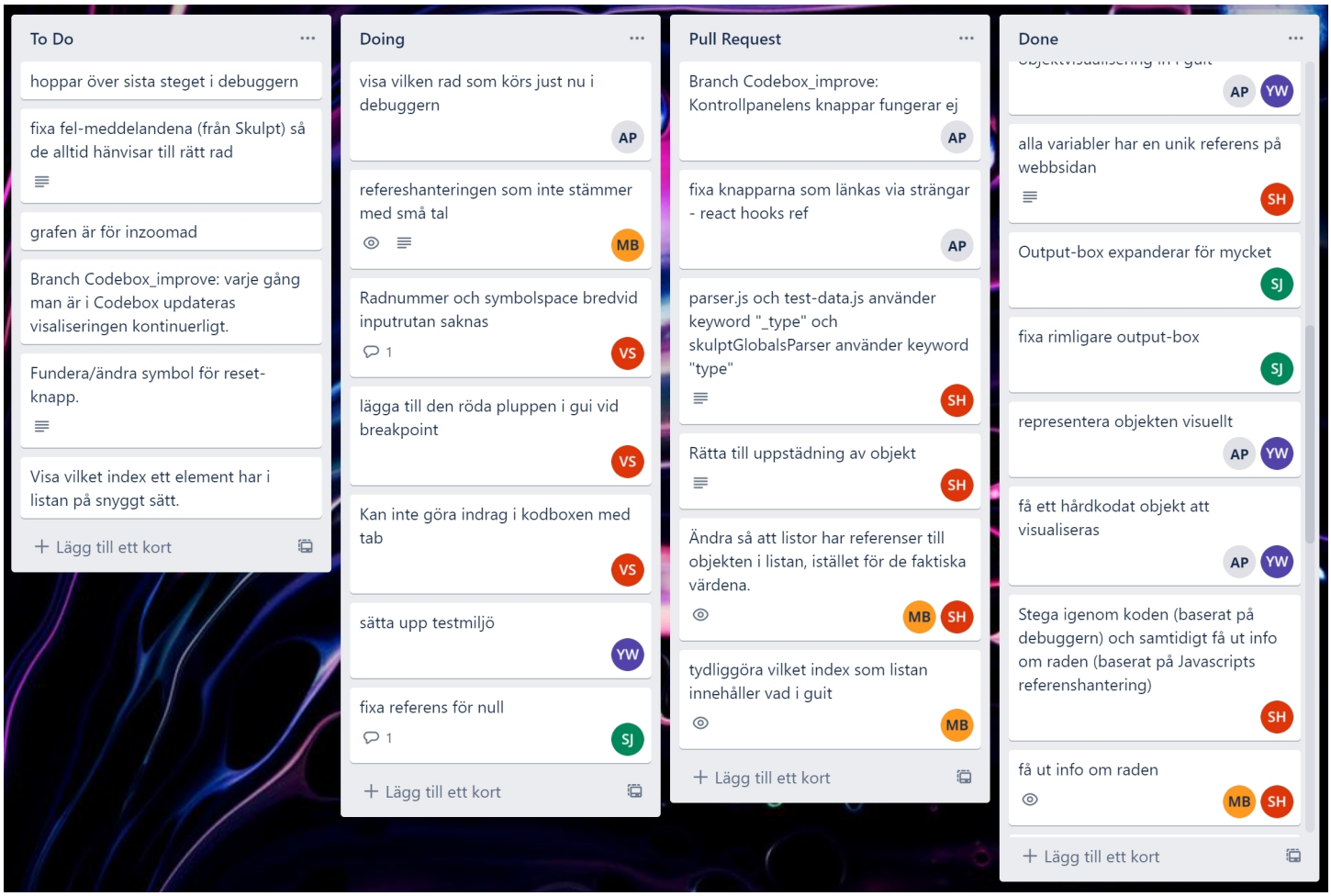
[27] World Health Organization, ”Disability,”. [Online]. Tillgänglig: https://www.who.int/health-topics/disability (hämtad: 2022-01-26).

[28] Funka, ”Datorhjälpmedel,”. [Online]. Tillgänglig: https://www.funka.com/design-for-alla/information-webb-och-it/datorhjalpmedel/ (hämtad: 2022-03-18).

[29] World Health Organization, ”Disability: What is e-accessibility?,” 2013. [Online]. Tillgänglig: https://www.who.int/news-room/questions-and-answers/item/what-is-e-accessibility (hämtad: 2022-01-26).
[30] Myndigheten för digital förvaltning, ”Om lagen om tillgänglighet till digital offentlig service,”. [Online]. Tillgänglig: https://www.digg.se/digital-tillganglighet/om-lagen#vanliga_fragor_och_svar1038 (hämtad: 2022-03-18).
[31] Myndigheten för digital förvaltning, ”Webbdirektivet – översikt,”. [Online]. Tillgänlig: https://webbriktlinjer.se/lagkrav/webbdirektivet/ (hämtad: 2022-03-18).
[32] *Accessibility requirements for ICT products and services*, EN 301 549 V3.2.1, European Telecommunications Standards Institute, European Committee for Standardization och European Electrotechnical Committee for Standardization, Belgien och Frankrike, 2021. Tillgänglig: https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf.
[33] World Wide Web Consortium, ”Web Content Accessibility Guidelines (WCAG) 2.1,” 2018. [Online]. Tillgänglig: https://www.w3.org/TR/WCAG21/ (hämtad: 2022-03-18).
[34] Python Tutor, 2022, [Mjukvara], USA: Philip Guo.
[35] Python Tutor Community, ”hcientist/OnlinePythonTutor,”. [Online]. Tillgänglig: https://github.com/hcientist/OnlinePythonTutor (hämtad: 2022-03-01).
[36] d3-graphviz, Version 4.1.1, [Mjukvara], d3-graphviz Community, 2022.
[37] CodeMirror, Version 5.65.2, [Mjukvara], CodeMirror Community, 2022.
[38] Queue-Fair, ”How does too much traffic crash a website?,” 2021. [Online]. Tillgänglig: https://queue-fair.com/how-traffic-can-crash-your-website (hämtad: 2022-04-19).
[39] IBM, ”Injection attacks,” 2021. [Online]. Tillgänglig: https://www.ibm.com/docs/en/snips/4.6.0?topic=categories-injection-attacks (hämtad: 2022-04-26).
[40] Pyodide Community, ”What is Pyodide?,”. [Online]. Tillgänglig: https://pyodide.org/en/stable/ (hämtad: 2022-05-01).
[41] Graphviz Community, ”Graphviz,” 2021. [Online]. Tillgänglig: https://graphviz.org/ (hämtad: 2022-04-04).
[42] Graphviz Community, ”DOT Language,” 2022. [Online]. Tillgänglig: https://graphviz.org/doc/info/lang.html (hämtad: 2022-04-19).
[43] react-graph-vis, Version 1.0.5, [Mjukvara], react-graph-vis Community, 2021.
[44] GitHub Pages, 2022, [Mjukvara]: GitHub.
[45] M. Rehkopf, ”What is continuous integration?,” *Atlassian*, [Online]. Tillgänglig: https://www.atlassian.com/continuous-delivery/continuous-integration (hämtad: 2022-02-19).
[46] Eslint Community, ”ESLint: Find and fix problems in your JavaScript code,”. [Online]. Tillgänglig: https://eslint.org/ (hämtad: 2022-02-19).
[47] Prettier Community, ”Prettier: opinionated code formatter,”. [Online]. Tillgänglig: https://prettier.io/ (hämtad: 2022-02-19).

[48] Box UK, "The seven principles of testing,". [Online] Tillgänglig: https://www.boxuk.com/insight/the-seven-principles-of-testing/ (hämtad: 2022-03-27).
[49] Chalmers tekniska högskola, "Tillgänglighetsredogörelse," 2021. [Online]. Tillgänglig: https://www.chalmers.se/sv/om-chalmers/om-webbplatsen/Sidor/tillganglighet.aspx (hämtad: 2022-03-18).
[50] Jam, Version 3.1.0, [Mjukvara]: Michael Amprimo, 2020.
[51] Contrast Checker, 2022, [Mjukvara], Logan, UT, USA: WebAIM.
[52] Meta Open Source, "React: A JavaScript library for building user interfaces," 2022. [Online]. Tillgänglig: https://reactjs.org/ (hämtad: 2022-03-08).
[53] A. Breman och A. Felländer, "Diginomics – nya ekonomiska drivkrafter," *Ekonomisk Debatt*, vol. 42, nr. 6, 2014. [Online]. Tillgänglig: https://www.nationalekonomi.se/sites/default/files/2014/10/42-6-abaf.pdf, Hämtad: 2022-03-16.
[54] J. Sturtz, "Dictionaries in Python," *Real Python*, [Online]. Tillgänglig: https://realpython.com/python-dicts/ (hämtad: 2022-04-21).
[55] Python Software Foundation, "Built-in Functions," 2022. [Online]. Tillgänglig: https://docs.python.org/3/library/functions.html#round (hämtad: 2022-04-08).
[56] T. Moss, "WCAG 2.0: The new W3C web accessibility guidelines evaluated," *Code Project*, [Online]. Nov. 26, 2006. Tillgänglig: https://www.codeproject.com/Articles/16536/WCAG-2-0-The-new-W3C-web-accessibility-guidelines, (hämtad: 2022-05-07).
[57] UserWay, "The Future of Web Content Accessibility Guidelines and WCAG 3.0,". [Online]. Tillgänglig: https://userway.org/blog/the-future-of-web-content-accessibility-guidelines-wcag30/, (hämtad: 2022-05-07).
[58] CodingBat, 2022, [Mjukvara], USA: Nick Parlante.

A
Scrumtavla



Varje vitt kort representerar en programmeringsuppgift. De fyra kolumnerna visar om uppgiften är planerad att göras, är påbörjad, väntar på att bli godkänd, eller är slutförd. De runda symbolerna i nedre höger hörn på korten representerar vem som tagit sig an uppgiften.

B
Prototyp


1 a = [1,2,3,4]
2 b = a
3
4 b.pop()
5 print(a)
6 print(b)
7
8
9
10
11
12
13
14
15
16
17
18
19
20
> [1,2,3]
>
a
1,2,3
b

C
Användartester 1.0

Placering av utskrift

- 3 av 5 hade rätt på vart terminalen befinner sig.
- 1 av 5 tyckte att terminalen var en av de stora rutorna, eftersom den riktiga terminalen uppfattades som en terminalemulator.

Placering av visualisering

- 5 av 5 hade rätt på var visualiseringen befinner sig.
- 1 av de som tidigare hade fel på terminalen rättade sig då frågan om visualiseringsrutan kom.

Stegning

- 4 av 5 hade inget problem att hitta ”steg”-knappen.
- 1 av 5 hade det lite svårt men hittade knappen till slut, beror på att hen inte har använt avbuggare tidigare.

Att starta om

- 2 av 5 hade inget problem med att starta om programmet.
- 1 av 5 tryckte inte på knappen utan kopierade koden och uppdaterade sidan.
- 2 av 5 tryckte på andra knappar (d.v.s. rensa brytpunkter) först innan de hittade ”stopp”-knappen.

Att exekvera hela koden

- 5 av 5 hade inget problem att exekvera hela koden.

Visualisering

- 5 av 5 hade inget problem med att beskriva visualiseringen av det enkla exemplet.

Felmeddelande

- 5 av 5 hade inget problem med att fixa felet.

Att lägga till en brytpunkt

- 3 av 5 lade till en brytpunkt utan problem.
- 1 av 5 lyckades lägga till en brytpunkt efter en tids förvirring.
- 1 av 5 visste inte vad en brytpunkt var och behövde förklaring för att komma vidare.

Att exekvera till brytpunkt

- 4 av 5 hade inget problem.
- 1 av 5 försökte att stega till brytpunkt först, och sedan tryckte hen på ”kör”-knappen.

Att redigera i koden under exekvering

- 1 av 5 förstod att det var den gamla koden som exekverades.
- 4 av 5 förstod inte varför utskriften inte stämde med den nya koden.

Förväntad utskrift från ett program med sidoeffekter

Resultatet från detta test påverkar inte vår produkts kvalitet.

- 2 av 5 svarade korrekt.

Att förstå sidoeffekter från visualiseringen

- 2 av 5 visste redan vad som kommer att hända.
- 2 av 5 lärde sig att det är referenser som tilldelas och inte en kopia.
- 1 av 5 hade lite svårt att förstå referenser, vilket kan bero på att hen har lärt sig C++ innan som behöver att pekare markeras explicit.

Informationssidan

- 5 av 5 tyckte att det såg bra ut.
- En användare ansåg att det kanske kan finnas en liten bild i hörnet.
- En annan användare märkte att koden inte var tillgänglig även om det står att den finns på GitHub.

Bugg

- En användare upptäckte att när for-loopar körs så visualiserade inget.
- En användare testade med ** operator och dictionary men inget hände.

Färgtema

- Tre användare kommenterade på det ljusa färgtemat och föredrog mörkt tema.
- En användare ansåg att metoderna bör ha en annan färg för att vara mer lättläsliga.

Övrigt

- En användare förstod inte att programmet inte exekverar den raden som stannas vid.
- En användare ansåg att när användaren går tillbaka från informationssidan till huvudsidan så ska koden finnas kvar.
- En användare tyckte att det bör finnas text till knapparna och en annan hade lite svårt att förstå vad knapparna innebar från början.
- En användare klagade på att knappen till informationssidan inte såg ut som en knapp. Hen ansåg att det bör tydliggöras att den är klickbar.
- En användare undrade om det är möjligt att använda sidan på mobilen.
- En användare försökte vid felmeddelande starta om koden genom att klicka på ”kör”-knappen. Hen förstod inte varför det inte fungerade och behövde hjälp med att förstå att man behövde trycka på ”stop”-knappen först.
- När ingen visualisering skapades så tryckte en användare flera gånger på ”kör”-knappen för hen trodde att inget exekverades. Utskrifter borde visas tydligare i terminalen?
- En användare kommenterade på att man borde placeras i mitten av grafen varje gång koden exekveras.
- En användare kommenterade att ”hjälp”-knapp för Python hade varit trevligt.

Användarna blev ombedda att välja ut en emoji per rubrik som sammanfattande deras upplevelse av respektive ämne:

knappar	
grafen	
about	
färger	

D
Användartester 2.0

Placering av utskrift

- 2 av 5 svarade att utskrifter sker i terminalen.
- 1 av 5 svarade att utskrifter sker i den stora rutan (visualiseringsrutan).
- 2 av 5 svarade istället vad ”b” kommer att innehålla då de fick fel frågan.

Placering av visualisering

- 4 av 5 hade rätt på att visualisering sker i den stora rutan.
- 1 av 5 svarade inte vart visualiseringen sker utan beskrev vad som kommer att visas i visualiseringsrutan.

Stegning

- 5 av 5 hade inget problem att hitta ”steg”-knappen.

Att starta om

- 2 av 5 hittade ”stopp”-knappen utan problem.
- 2 av 5 tryckte först på knappen för att rensa alla brytpunkter.
  - En användare upplevde att det var svårt att se skillnad mellan ”stop”-knappen och knappen för att rensa alla brytpunkter om man har dålig syn. Det borde vara tydligare ikoner.
- 1 av 5 uppdaterade sidan istället för att trycka på någon knapp.

Att exekvera hela koden

- 5 av 5 hade inget problem med att exekvera hela koden.

Att beskriva visualisering

- 3 av 5 hade inget problem med att beskriva visualiseringen.
- 2 av 5 förstod inte vad 0 (index) betyder.

Att lägga till en brytpunkt

- 5 av 5 lade till en brytpunkt utan problem.

Att exekvera till brytpunkt

- 4 av 5 gjorde rätt.
- 1 av 5 stegade fram till raden med brytpunkt.

Att ta bort sista rad när kodrutan är låst

- 4 av 5 insåg att kodrutan var låst och tyckte att det kändes bra.
- 1 av 5 kommenterade att ”kör”-knappen borde ändra sig till en annan ikon för att visa att koden exekveras just nu.

Felmeddelande

- 4 av 5 hade inget problem med att fixa felet.
- 1 av 5 behövde lite hjälp men fixade felet till slut.

Att välja ett nytt kodexemplet

- 5 av 5 hade inget problem.

Förväntad utskrift från ett program med sidoeffekter

- 3 av 5 svarade korrekt.
  - En användare kommenterade på att visualiseringen hjälpte för att besvara frågan.
- 2 av 5 svarade men var inte helt korrekt.

Att tolka döda referenser och röda pilar

- 3 av 5 förstod visualiseringen och vad den röda pilen innebar.
- 1 av 5 förstod inte om det objekt som den röda pilen pekar på kvarstod i minnet eller om det bara betyder att det inte har något referens.
- 1 av 5 tyckte att är det är otydligt vad de två olika pilarna betyder och föreslog att det kan finnas ett fält som beskriver vad respektive pil innebär.

Informationssidan

- 2 av 5 ansåg att inget saknades på informationssidan.
- En användare ansåg att det kanske bör tydliggöras vilka datatyper som inte stöds av LearnyPy (kanske bara några exempel). Just nu känns det som att det

stöder alla datatyper. Användaren tycker att även någon kontaktinformation är bra att ha.
- En användare saknade en förklaring av programmet, samt grafens komponenter.
- En användare kommenterade på att skrollningslisten visas inte på Opera eller Safari om man inte försöker skrolla.

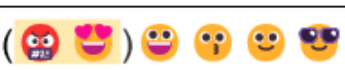
Bugg

- Skroll visades i terminalen även om terminalen var tom.
- Ibland kvarstod felmeddelanden även om problemet var åtgärdat.

Övrigt

- En använde trodde att bakgrunden på informationssidan var olika flikar som går att klicka på.
- En användare ansåg att knapparna kan förbättras. Hen tyckte att skillnaden mellan ”kör”- och ”steg”-knappen inte var trivial och det kanske borde vara en ”loop”-ikon för ”kör”-knappen istället.
- En användare upptäckte att när sidan uppdaterades så ändrades färgtemat alltid tillbaka till ljust. Hen ansåg att det borde fixas webbkakor så att det sparas. Hen föreslog även en knapp för att tömma kodrutan.
- En användare ansåg att koden i kodrutan borde finnas kvar när man går tillbaka från en annan sida. Användaren föreslog att programmet bör visa vart man befinner sig just nu. Hen tyckte att två fingrar upp eller ner i grafen borde “skrolla” och inte zooma.

Användarna blev ombedda att välja ut en emoji per rubrik som sammanfattande deras upplevelse av respektive ämne:

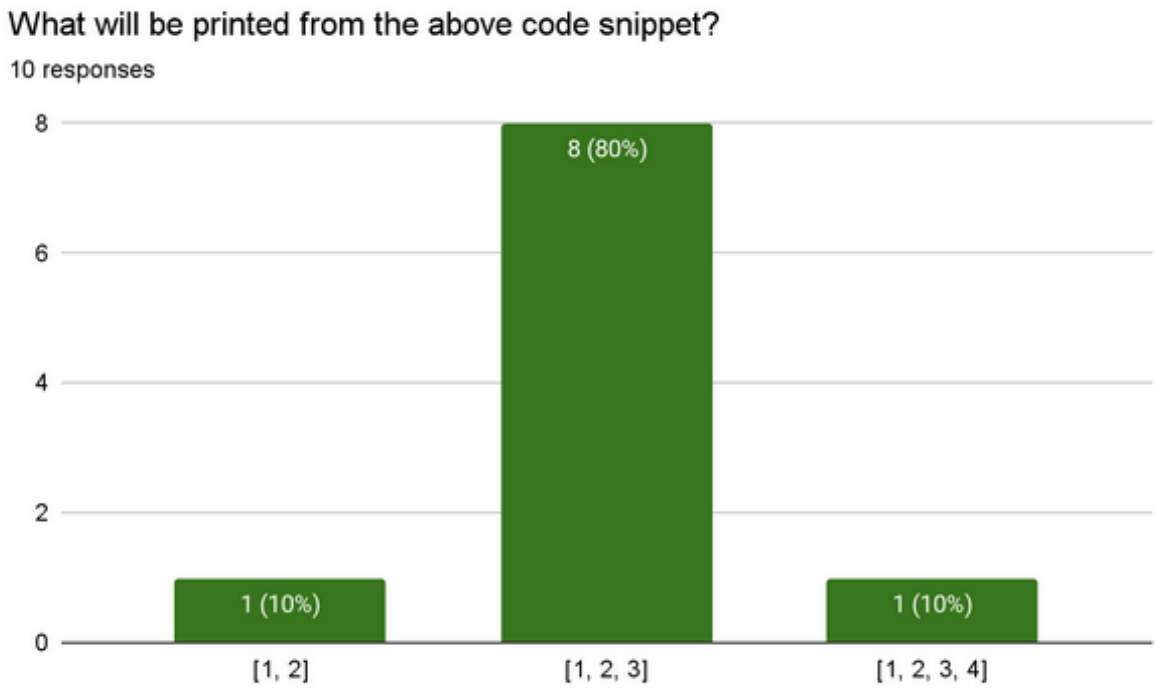
knappar	
grafen	
about	
färger	

E
Resultat av formulären

Observe the following code snippet. The append() method adds an item to the end of the list.

```
a = [ 1, 2 ]
b = a
b.append(3)
print(a)
```

What will be printed from the above code snippet?



Please go to `https://learnypy.netlify.app/` and run the previous code snippet. Did the output meet your expectations? If not, did the visualisation clear any misconceptions that you had?

- 10 out of 11 users think the output meets their expectations.

- 1 user thinks the visualisation was hard to understand. Did not provide a reason.

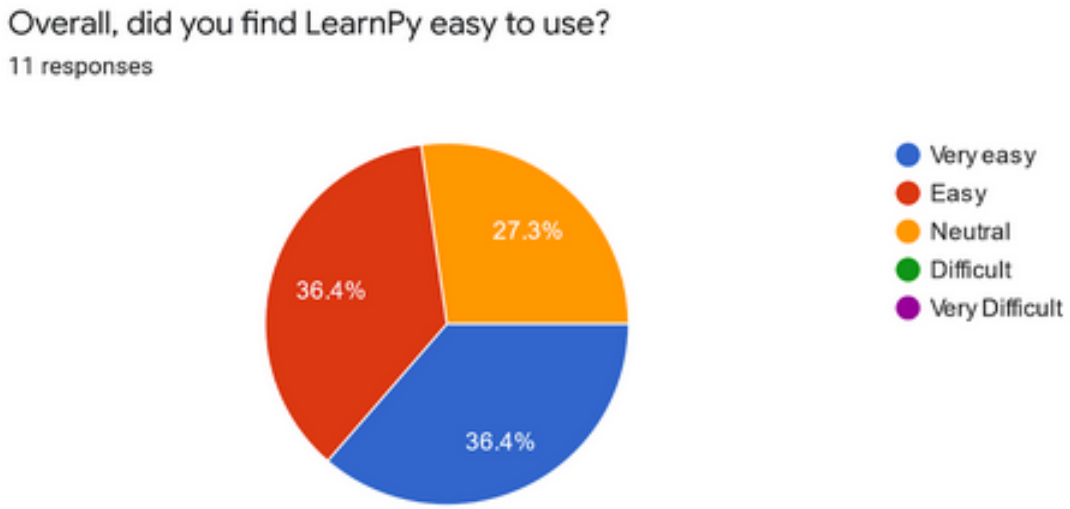
Please enter the following code snippet without modifying it:

```
import random
rand = random.randint(0, 9)
if rand = 7:
  print('Lucky Seven!')
else:
  print('Not so lucky')
```

Was it easy to identify and resolve the error?

- 8 out of 11 users think that it was easy to identify and resolve the error.
- 1 out of 11 users thinks it was a bit hard but manageable.
- 2 out of 11 users think it doesn't make it easier to find the error.
- 2 users commented that the error message is uninformative and it would be more helpful if the error message was more detailed.
- 1 user commented that the error message points to the wrong line.

Overall, did you find LearnyPy easy to use?



What did you think of LearnPy? What can be improved? (optional)

- 1 user commented that it helps learning basic python logic, but does not help much when it comes to functions and classes.
- 1 user commented that it was difficult to use on the mobile browser, but it seems to work fine.
- 2 users left comments saying it was helpful.

F
Tillgänglighetsredogörelse

Accessibility of LearnPy

We want as many people as possible to be able to use it, and this document describes how https://learnypy.netlify.app/ complies with the accessibility regulations, any known accessibility issues, and how you can report problems so that we can fix them.

How Accessible Is the Website?

We know some parts of this website are not fully accessible. See the section on non-accessible content below for more information.

What to Do if You Can Not Access Parts of This Website?

If you need content from this website that is not accessible for you, but is not within the scope of the accessibility regulations as described below, please report it to the issues page.

Reporting Accessibility Problems With This Website

We're always looking to improve the accessibility of this website. If you find any problems that aren't listed on this page or if we're not meeting the requirements of the accessibility regulations, let us know about the problem by reporting it to the issues page.

Enforcement Procedure

The Agency for Digital Government is responsible for enforcing the accessibility regulations. If you are not happy with how we respond to your complaint about web accessibility or your request to make content accessible, submit a complaint to the Agency for Digital Government.

Technical Information About This Website's Accessibility

This website is partially compliant with level AA in the standard Web Content Accessibility Guidelines version 2.1, due to the non-compliances listed below.

Non-accessible Content

The content described below is, in one way or another, not fully accessible.

- Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element (WCAG 2.1, criteria 1.4.1)
- In content implemented using markup languages that support the following text style properties, no loss of content or functionality occurs by setting all of the following and by changing no other style property (WCAG 2.1, criteria 1.4.12):
  - Line height (line spacing) to at least 1.5 times the font size.
  - Spacing following paragraphs to at least 2 times the font size.
  - Letter spacing (tracking) to at least 0.12 times the font size.
  - Word spacing to at least 0.16 times the font size.
- Make all functionality available from a keyboard (WCAG 2.1, criteria 2.1.1)
- More than one way is available to locate a Web page within a set of Web pages except where the Web Page is the result of, or a step in, a process (WCAG 2.1, criteria 2.4.5)

How We Tested This Website

We have conducted a self-assessment (internal testing) of LearnPy.

The website has not been officially published yet.

The statement was last updated on 2022-05-06.