Sten-Saks-Papir

Programmering B

Jacob Søgaard - 2g

ZBC Slagelse HTX d. 30/12 - 2025

Øvrige gruppemedlemmer: Navn og navn

Vejleder: Navn

Abstract

Denne synopsis dokumenterer en implementation af spillet sten-saks-papir. Programmet er skrevet i Python og det afvikles vha. kommandolinjen. Programmet er opbygget af to primære funktioner udover hovedfunktionen hvor sekvenseringen af programmet er defineret. Vi har desuden udarbejdet en datastruktur til at repræsentere spillernes valg vha. en `IntEnum` klasse.

Vi anvender indbyggede biblioteker og funktioner bl.a. til at modtage input, udskrive output og generere tilfældige tal. Synopsen indeholder en overordnet beskrivelse af alle funktioner, samt en detaljeret beskrivelse af de to primære funktioner.

Vi har udført funktionelle test af udvalgte dele af programmet samt hele programmet samlet. Vi kan vha. de afsluttende test konkludere, at programmet opfører sig som forventet ved pæne brugerinput. Det er dermed muligt at spille en omgang af sten-saks-papir mod en computermodstander.

Indholdsfortegnelse

1 Indledning

I dette projekt har vi undersøgt hvordan der kan fremstilles et simpelt program i Python til at afvikle det klassiske spil "Sten/saks/papir". Spillet skal styres vha. kommandolinjer i en terminal.

Programmet er udarbejdet i en gruppe på 3 personer. Person XXX har haft ansvaret for generering af computerens valg, mens Person YYY har haft ansvar for input/output. Person ZZZ har haft ansvar for overordnet strukturering af koden samt hvordan vinderen bestemmes.

Resten af synopsen er struktureret som følger: I metodeafsnittet relateres udviklingsarbejdet til programmeringsfagets metoder. Programmets funktionalitet og de forskellige indtastningsmuligheder gennemgås i det efterfølgende afsnit. Herefter følger synopsens hovedafsnit med dokumentation af selve programmet. Her gives først et overblik over programmets overordnede opbygning hvorefter funktionen `display_result` og derefter `get_player_choice` er beskrevet i detaljer. I testafsnittet beskrives både funktionelle test og brugertest. Selve synopsen rundes til sidst af med en konklusion på de væsentligste problemstillinger som præsenteret i problemformuleringen herunder.

1.1 Problemformulering

For at implementere spillet "Sten/saks/papir" er der blevet lagt vægt på følgende problemstillinger:

- Hvordan repræsenteres computerens og brugerens valg i passende datatyper?
- Hvordan genereres et tilfældigt udfald og hvordan bestemmes vinderen?
- Hvordan modtages brugerens input og hvordan vises vinderen?

2 Metode

Vores overordnede metode til udviklingen af programmet har været trinvis forbedring. I starten af projektet opstillede vi versionbeskrivelser for tre versioner af produktet som fremgår af vores versionsplan i Tabel 1. Her svarer hver version til en udvidelse af koden. Som det også fremgår af tabellen, har vi bl.a. planlagt arbejdsprocessen ved, at strukturere koden med funktionsstubbe.

Tabel 1: Versionsplan.

Version	Beskrivelse
1	Overordnet struktur vha. funktionsstubbe.
2	Implementation: Tilfældig computervalg.
3	Implementation: Modtag input og sammenlign.

Undervejs i projektet har vi som en del af den trinvise forbedring evalueret vores foreløbige produkt og tilpasset beskrivelserne. De endelige versionsbeskrivelser fremgår af versionshistorikken i Tabel 2. En stor del af version 3 i den endelige versionshistorik var omstrukturering af koden. Denne omstrukturering indbefattede bl.a. at initialiseringen af tilfældighedsgeneratoren blev flyttet til sin egen funktion for at styrke læsbarheden af koden.

Tabel 2: Versionshistorik (realiserede versioner).

Version	Beskrivelse
1	Overordnet struktur vha. funktionsstubbe.
2	Implementation: Tilfældig computervalg.
3	Implementation: Modtag input. Omstrukturering af kode.
4	Implementation: Sammenlign valg og bestem vinder.

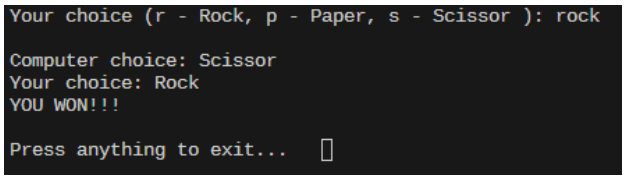
I de fleste versioner har der også været forfinelse undervejs. Fx blev der opdaget en fejl i logikken til at bestemme vinderen af spillet når spillerens værdi svarede til heltallet 0. Det blev rettet til den version der er dokumenteret i Sektion 4.2.

Som en del af vores udviklingsarbejde har vi brugt git som versionsstyring. Under udviklingsarbejdet oprettede vi grene til hver version og kildekoden for de forskellige versioner fremgår derfor af de tilhørende grene i vores repository [1].

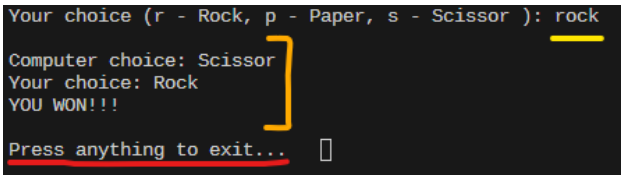
3 Funktionsbeskrivelse

Programmet afvikles i terminalen vha. kommandolinjer. Når programmet køres vil brugeren først blive bedt om at foretage sit valg mellem sten, papir og saks. Det forventes at brugeren indtaster sit valg i form af en tekststreng hvor valget enten kan skrives på engelsk eller angives vha. første bogstav i ordet. De lovlige input til sten er fx "r", "rock" eller et af de to forrige med store bogstaver (på vilkårlige placeringer i ordet).

Efter at brugeren har indtastet sit valg foretager programmet et tilfældigt valg af de tre mulige, udregner hvem der har vundet og viser det til brugeren. Til sidst kan brugeren trykke på enter-tasten for at afslutte programmet eller lukke vinduet. Figur 1 viser programmets output efter brugeren har valgt papir og på Figur 2 er de forskellig trin markeret. Gul: brugerinput, orange: visning af resultater og rød: afsluttende tekst.



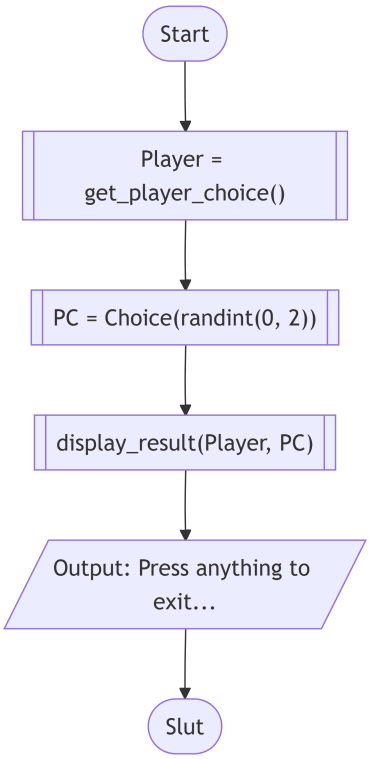
Figur 1: Programoutput efter indtastning: "rock"



Figur 2: Programoutput med markeringer.

4 Dokumentation

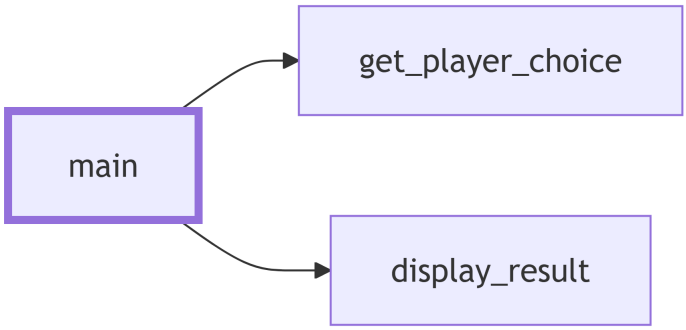
I dette afsnit beskrives implementering først overordnet og derefter beskrives udvalgte dele af koden mere detaljeret i underafsnit. Programmets overordnede opbygning er vist vha. et rutediagram i Figur 3. Selvom programmet kun kalder hver funktion en gang er der gjort brug af funktioner for at strukturere koden og gøre den mere læsbar. Hovedfunktionen `main` indeholder variabler for computerens og brugerens valg og kalder de øvrige funktioner i sekvens.



Figur 3: Rutediagram. Pga. layout er betingelser markeret ved 6-kantede figurer i stedet for den sædvanlige rombe.

Programmet er opdelt i to primære funktioner som kaldes af `main`, som det fremgår af Figur 4. En kort beskrivelse

af programmets funktioner fremgår Tabel 3. Funktionen `display_result` og `get_player_choice` er detaljeret beskrevet i henholdsvis Sektion 4.2 og Sektion 4.1.



Figur 4: Funktionskald i `main`.

Tabel 3: Oversigt over programmets funktioner

Funktion	Beskrivelse
`int main()`	Hovedfunktionen med overordnet struktur.
`int get_computer_choice()`	Udvælger tilfældigt et valg for computeren.
`void display_result(int P, int C)`	Bestemmer vinderen og viser resultatet.

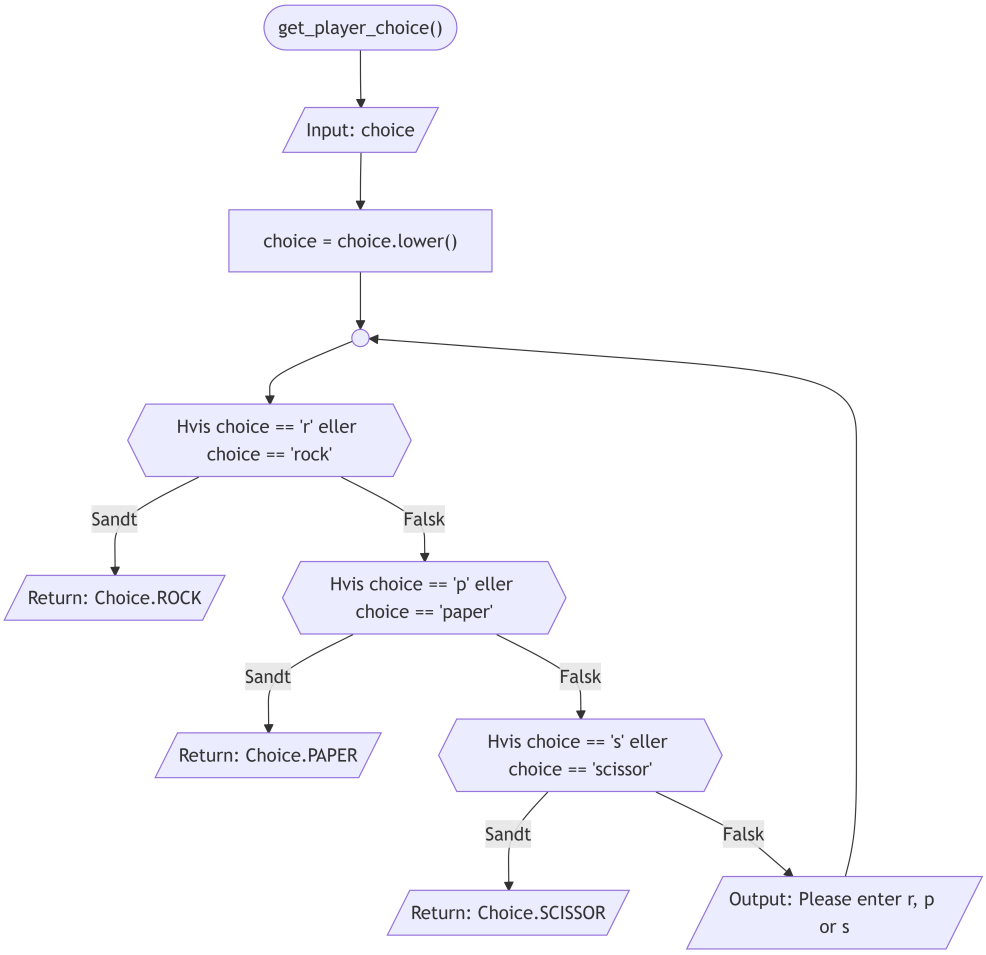
Der benyttes 2 moduler fra Pythons standardbibliotek: `random` benyttes til at generere tilfældige heltal og `Enum` bruges til at repræsentere et valg i spillet. Standardfunktionerne `input` og `print` benyttes til hhv. input og output. Input fra brugeren modtages hovedsageligt i funktionen `get_player_choice` og forklares i Sektion 4.1. I `main` bruges `input` dog også afslutningsvis til at holde vinduet åbent, hvis programmet senere skulle kunne køres som en selvstændig applikation. Fra `random` importeres funktionen `randint` som bruges til at generere et tilfældigt heltal mellem 0 og 2. Dette kan direkte oversættes til et valg i spillet.

Repræsentationen af valg i spillet har gennemgået flere iterationer i udviklingen af programmet. Til at starte med brugte vi først heltal til at repræsentere spillerens og computerens valg. Senere skiftede vi til datatypen `Enum`, da et valg kun har 3 muligheder. Efter at have undersøgt [2] nærmere skiftede vi til datatypen `IntEnum` således at et valg også kan opfattes som et heltal, hvilket gør sammenligning mellem to valg nemmere. Denne sammenligning er nærmere beskrevet i dokumentationen af `display_result` i Sektion 4.2. På denne måde har vi klart defineret at der kun er tre lovlige værdier samtidig med at vi har mulighed for at bruge heltal i sammenligninger.

4.1 Funktionen get_player_choice

Funktionen `get_player_choice` sørger for at brugeren indtaster sit valg og returnerer det som en `Choice`. Kildekoden for funktionen er medtaget herunder (uden kommentarer) og tilhørende rutediagram fremgår af Figur 5.

```
1  def get_player_choice() -> Choice:
2      while True:
3          choice = input("Your choice (r - Rock, p - Paper, s - Scissor ): ")
4          match choice.lower():
5              case "r" | "rock":
6                  return Choice.ROCK
7              case "p" | "paper":
8                  return Choice.PAPER
9              case "s" | "scissor":
10                 return Choice.SCISSOR
11             case _:
12                 print("Please enter r, p or s")
```



Figur 5: Rutediagram for funktionen get_player_choice.

I funktionsignaturen fremgår det at funktionen ikke har nogen parametre og returnerer en variabel af datatypen `Choice`. I kodelinje 2 startes en uendelig løkke vha. `while True`. Dermed sikrer vi at der først returneres fra funktionen når en af de tre betingelser for et valg i spillet er opfyldt.

På linje 3 bedes brugeren om at indtaste sit valg som gemmes som en tekststreng. På næste linje konverteres valget til små bogstaver vha. metoden `lower` og det er denne udgave af tekststrengen der bruges til `match`. På den måde er det ligegyldigt om brugeren indtaster store eller små bogstaver.

`Match` er en relativ ny feature i Python og er introduceret i version 3.10 [3]. Under `match` er der skrevet tre `case`-blokke som matcher på et enkelt bogstav for valget eller på hele ordet fx "r" eller "rock". Det gøres vha. `|`-operatoren i sætningen med `case`. Hvis en af de tre muligheder for et valg er opfyldt returneres det tilsvarende valg som en `Choice` og da der returneres ud af funktionen brydes den uendelige løkke dermed også.

Den sidste case på linje 11 matcher på alle andre muligheder vha. `_` og beder brugeren om at indtaste et lovligt input. I dette tilfælde fortsætter løkken og der bedes dermed igen om input fra brugeren i linje 3.

4.2 Funktionen display_result

Funktionen `display_result` afgør resultatet af spillerens og computerens valg og udskriver derefter resultatet. Den er derfor en central del af programmet og forklares detaljeret i dette afsnit. Kildekoden for funktionen er medtaget herunder (uden kommentarer) og tilhørende rutediagram fremgår af Figur 6.

```
1  def display_result(player: Choice, computer: Choice):
2      print(f"\nComputer choice: {computer}")
3      print(f"Your choice: {player}")
4
5      if player == (computer + 1) % 3 or (player == 0 and computer == 2):
6          print("YOU WON!!!")
7      elif player != computer:
8          print("YOU LOST!!!")
9      else:
10         print("IT'S A TIE!")
```

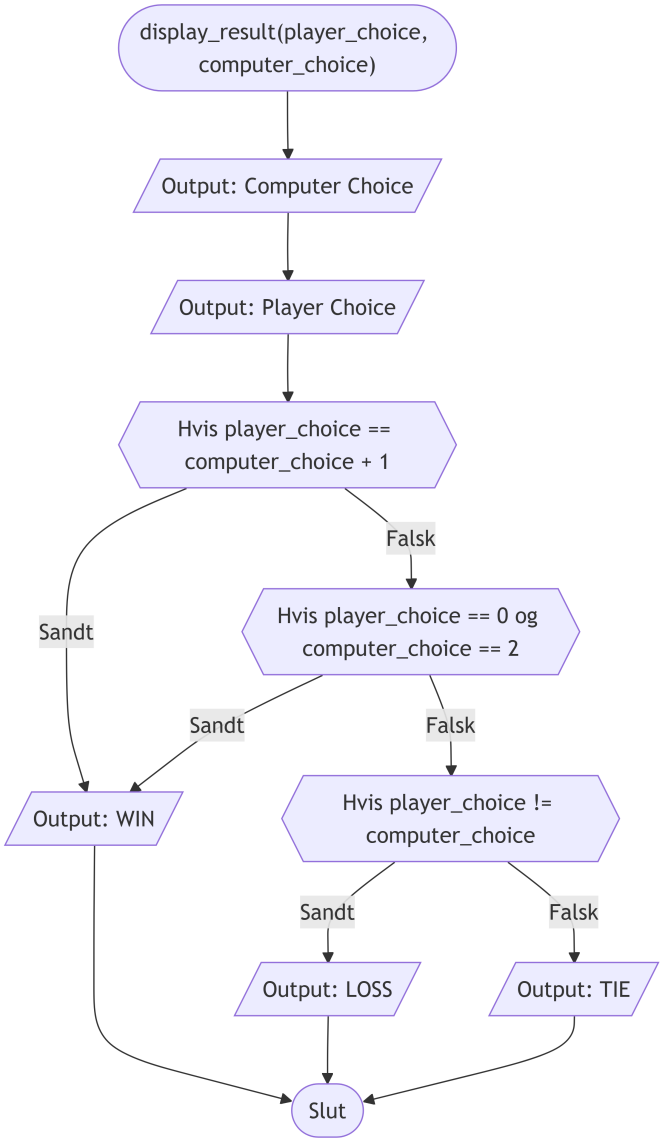
Funktionen har to parametre som det fremgår af funktionssignaturen i kodelinje 1. Begge parametre er af datatypen `Choice` som er en `IntEnum`. Parameteren `player` repræsenterer brugerens valg og `computer` repræsenterer computerens valg. Pga. datatypen kan parametrene kun antage tre forskellige værdier: `Choice.ROCK`, `Choice.SCISSOR` eller `Choice.PAPER`. Værdierne gemmes som tre forskellige heltal og datatypen kan også bruges i alle sammenhæng hvor heltal kan bruges pga. egenskaberne for en `IntEnum` [2]. Funktionen returnerer ingen værdi.

På linje 2 udskrives først et linjeskift vha. `\n`, derefter strengen "Computer choice:" efterfulgt af strengen som repræsenterer computerens valg. I `Choice` har vi overlæsset metoden `__str__` således at datatypen nemt kan konverteres til en tekststreng. Det udnyttes her hvor variablen indsættes i en formatteret tekststreng. Dernæst udskrives brugerens valg på lignende måde på næste linje.

Fra linje 5 til 10 sammenlignes de to valg i en kontrolstruktur der er opbygget af tre `if-else`-sætninger. Forgreningerne fremgår også meget tydeligt i rutediagrammet vist i Figur 6. Datatypen `Choices` er opbygget således at $n + 1$ slår n når værdierne betragtes som heltal. Derfor tjekkes i første betingelse om brugerens valg `player` er lig computerens valg adderet med 1 dvs. `computer + 1`. Undtagelsen er når computeren har valgt det sidste element. I det tilfælde har brugeren vundet hvis han/hun har valgt det første element. Det kædes derfor sammen med første betingelse for at spillerne har vundet vha. `or` til sidst i linje 7.

Hvis brugeren ikke er fundet som vinder tjekkes der på linje 7 om brugerens valg overhovedet er forskelligt fra computerens. Hvis det er tilfældet må computeren have vundet, idet betingelserne for en sejr for brugeren ikke var opfyldt. Sidste mulighed er at de to valg er ens, hvilket den sidste `else` tager højde for på linje 9. I alle ovenstående tilfælde udskrives det relevante resultat.

Et alternativ til forgreningerne kunne være `match` som der er blevet benyttet i funktionen `get_player_choice`. Det ville dog umiddelbart kræve at der først blev udregnet et resultat (fx de to valg fratrukket hinanden) som ville kunne bruges til at matche på. Vi vurderede derfor at logikken var nemmere at forstå med den nuværende løsning med `if-else`-sætninger.



Figur 6: Rutediagram for funktionen display_result.

5 Test

5.1 Funktionelle test

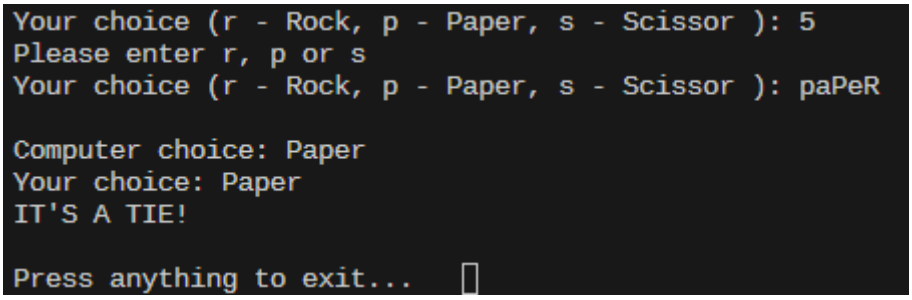
Vi har i løbet af udviklingen testet forskellige dele af programmet. Fx opdagede vi en fejl i funktionen `display_result` hvor spillet blev opfattet som uafgjort, hvis spilleren valgte sten og computeren valgte saks. Det blev rettet ved at tilføje en ekstra betingelse til at tjekke for dette specieltilfælde. Vi opdagede fejlen ved at teste funktionen gentagne gange med forskellige input. Vi overvejede at lave automatiske test til netop denne funktion. Det kunne vi have gjort ved at skrive testkode der kaldte funktionen med alle kombinationer af spillervalg og computervalg for derefter at tjekke resultatet. Vi endte dog med kun at teste funktionen manuelt.

Programmet er blevet testet med forskellige udgaver af lovlige og ulovlige input som det fremgår af Tabel 4. Derudover er afgørelsen for sejr blevet testet ved at afprøve programmet således at de tre mulige udfald er opnået minimum 1 gang. I alle tilfældene med lovlige input var det forventede output en korrekt betegnelse for både brugerens valg og computerens valg samt en korrekt afgørelse af vinderen. Alle disse tests blev gennemført med success.

Tabel 4: Tabel over funktionelle tests.

Input	Forventet output	Faktisk output
Tom tekststreng	Intet.	Programmet beder om nyt lovligt input.
“r”	Spillet afvikles normalt.	Spillet afvikles normalt.
“paPeR”	Spillet afvikles normalt.	Spillet afvikles normalt.
5	Programmet beder om nyt lovligt input.	Programmet beder om nyt lovligt input.

Et eksempel på en af disse test fremgår af Figur 7 hvor brugeren først prøver at taste “5” for derefter at give et lovligt input. Ud fra testene har vi bedømt at programmet generelt fungerer som forventet.



Figur 7: Test med brugerinput “5”.

5.2 Brugertest

Vi har udført brugertest af programmet på tre klassekammerater. To af testpersonerne havde nemt ved at bruge programmet uden instruktion, mens den sidste ikke var vant til at bruge terminalen og derfor skulle bruge lidt hjælp til anvendelsen. Alle tre kunne nemt regne ud hvad formålet med programmet var, selvom det egentlig ikke fremgår af beskrivelsen i terminalen. Alle tre savnede muligheden for at spille flere omgange og en enkelt savnede også en grafisk brugerflade.

6 Konklusion

I denne synopsis har jeg dokumenteret implementationen af vores program til Sten-Saks-Papir. Programmet er skrevet i `Python` og det gør det muligt at spille sten-saks-papir mod computeren vha. kommandolinjen.

Jeg har beskrevet hvordan valgene i spillet bliver repræsenteret vha. lokale variabler i hovedfunktionen i form af en `IntEnum` klasse.

Jeg har dokumenteret hvordan der modtages brugerinput og genereres tilfældige valg for computeren vha. funktioner fra standard `Python` moduler. Funktionen som bestemmer vinderen og viser denne information til brugeren er blevet grundigt dokumenteret.

De udførte test viser at programmet fungerer som forventede ved både lovlige og ulovlige input.

Hvis der skulle arbejdes videre med dette projekt, så ville det være oplagt tilføje en hovedløkke til spillet. Den skulle gøre det muligt for brugeren at spille flere runder og evt. med pointgivning undervejs. Man kunne også tilføje en pænere brugerflade vha. et grafisk bibliotek som `Pygame` eller `Tkinter`.

7 Referencer

[1] P. Parker, A. And, og J. Søgaard, "Repository for Projekt Sten-Saks-Papir". 7. august 2025. Tilgængelig hos: http://example.com/bruger/repo

[2] "enum — Support for enumerations", i *Python 3 documentation*, The Python Software Foundation, 2025. Tilgængelig hos: https://docs.python.org/3/library/enum.html. [Set: 1. marts 2025]

[3] "4. More Control Flow Tools", i *The Python Tutorial*, The Python Software Foundation, 2025. Tilgængelig hos: https://docs.python.org/3/tutorial/controlflow.html. [Set: 1. marts 2025]

A Bilag

A.1 Kildekode

```python
from random import randint
from enum import IntEnum


class Choice(IntEnum):
    ROCK = 0
    PAPER = 1
    SCISSOR = 2

    def __str__(self):
        return self.name.capitalize()


def get_player_choice() -> Choice:
    """Anmod brugeren om at vælge mellem rock, paper og scissor."""
    while True:
        choice = input("Your choice (r - Rock, p - Paper, s - Scissor ): ")
        match choice.lower():
            case "r" | "rock":
                return Choice.ROCK
            case "p" | "paper":
                return Choice.PAPER
            case "s" | "scissor":
                return Choice.SCISSOR
            case _:
                print("Please enter r, p or s")


def display_result(player: Choice, computer: Choice):
    """Udregn vinder og vis resultatet.

    ### Parametre:
        - player (Choice): Spillerens valg.
        - computer (Choice): Computerens valg.
    """
    print(f"\nComputer choice: {computer}")
    print(f"Your choice: {player}")

    if player == (computer + 1) % 3 or (player == 0 and computer == 2):
        print("YOU WON!!!")
    elif player != computer:
        print("YOU LOST!!!")
    else:
        print("IT'S A TIE!")


def main():
    """Hovedfunktion."""

    player = get_player_choice()
    pc = Choice(randint(0, 2))
    display_result(player, pc)
    input("\nPress anything to exit...   ")


```

```
56 if __name__ == "__main__":
57     main()
```