Teorioppgave 1 – Exception:

Exceptions betyr unntak, men oppfører seg mer som forventinger. Exceptions brukes ofte for å gi tilbakemeldinger på forventede feilmeldinger. En typisk exception kan være «ValueError». Koden som kan utløse en exception, plasseres i try-blokken, og koden som skal håndtere exceptionen plasseres i except-blokken. Man kan ha flere except-blokker for å håndtere ulike typer exceptions på forskjellige måter. Dette lar deg ta spesifikke tiltak basert på hvilken type exception som oppstår. Under er et eksempel på bruken av «ValueError» hvor den slår ut hvis brukeren ikke skriver inn et heltall:

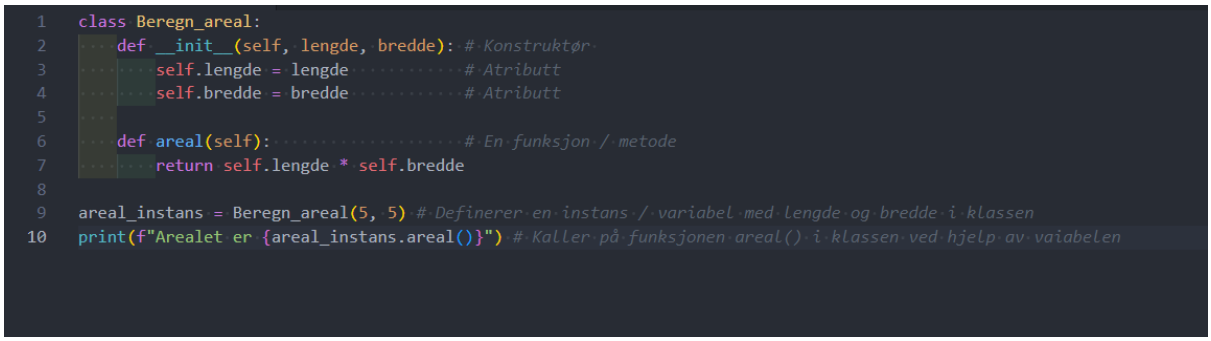
```
while True:
    try:
        input = int(input("Skriv et heltall: "))
    except ValueError:
        print("Du må skrive et heltall")
        continue
    else:
        print("Du har skrevet", input)
        break
```

Noen grunner til å bruke except, er feilsøking, for å sikre at programmet ikke krasjer når det oppstår feil, og for retting av feil i koden.

Teorioppgave 2 – Klasse:

En klasse en blokk med kode som brukes for å opprette objekter. Objekter brukes for å kalle på en funksjon/metode i klassen. En klasse skal alltid ha et navn som starter med stor forbokstav. Dette gjelder alle kodespråk, og er noe som gjør klassene lett gjenkjennelige. Videre har en klasse en konstruktør (__init__) som oppretter en ny instans av klassen. Den tar inn parametere som skal brukes i klassen. I dette tilfellet var det «lengde» og «bredde». «Self» skal alltid være med i klasser, ettersom det er den som brukes til å definere attributtene. Videre i klassen finner man metoder / funksjoner. Disse metodene tar ingen ekstra parametere, og brukes kun til å utføre oppgaver som er spesifikke for objekter av gjeldene klasse. Utenfor klassen kan man også lage variabler eller instanser av klassen, slik at det blir mer oversiktlig. Til slutt må man kalle på både klassen/variabelen og funksjonen man vil bruke innenfor

klassen. Utsnittet under viser en fullstendig klasse hvor det også kalles på en metode fra klassen.



```python
class Beregn_areal:
    def __init__(self, lengde, bredde): # Konstruktør
        self.lengde = lengde            # Atributt
        self.bredde = bredde            # Atributt

    def areal(self):                    # En funksjon / metode
        return self.lengde * self.bredde

areal_instans = Beregn_areal(5, 5) # Definerer en instans / variabel med lengde og bredde i klassen
print(f"Arealet er {areal_instans.areal()}") # Kaller på funksjonen areal() i klassen ved hjelp av vaiabelen
```

Teorioppgave 3 – Objekt:

I forrige oppgave ble det nevnt instans og variabel. En bedre terminologi på det er faktisk et «objekt». Disse objektene ligger ikke inne i klassen, men er heller noe man oppretter utenfor og styrer utfallet av hva som blir returnert i en eventuell «print». Hvis vi igjen ser på kodeeksempelet fra forrige oppgave, ser vi at linje 9 inneholder et objekt. Objektet ble navngitt «areal_instans», men kunne blitt navngitt noe helt annet. Det har egentlig ingenting å si. Å opprette et objekt og gi det et rimelig navn gjør det lettere og mer oversiktlig å bruke det senere eller flere ganger. Vi ser også objektet definerer de to pålagte parameterne som i dette tilfellet ble 5 og 5. Man kan lage så mange objekter man vil, da helst med forskjellige verdier og navn, ellers risikerer man å overskrive det man allerede har definert.

Programmeringsoppgave 2 - Dokumentasjon

Oppgavevalg

Da jeg fikk utdelt oppgaven og fikk opplyst at det var to fremgangsmåter var jeg først innstilt på at jeg kun skulle gå for «fremgangsmåte 1». Tingen var at jeg aldri hadde spilt Blackjack, og viste egentlig at poenget med spillet var å få 21. Jeg så meg derfor nødt til å starte med «fremgangsmåte 2». Jeg valgte også å kode på engelsk, mest for øving.

Oppstart:

Jeg lastet modulen som innehold alle kortene, kortverdiene og de ulike funksjonene. Jeg leste gjennom hele oppgaven og måtte til slutt spørre Chat GPT om en kjapp introduksjon til hva blackjack var og hvordan det skulle spilles. Etter å ha forstått meg litt mer på blackjack begynte jeg å lage et deck med tilfeldige kort, hvor jeg egentlig lånte funksjonen fra modulen. Deretter lagde jeg to tomme lister, én for spilleren (meg) og en for dealeren. Jeg lagde deretter en funksjon som skulle dele ut kort til disse listene, spilleren og dealeren. Jeg tok også to parametere i funksjonen, «player_amount» og «dealer_amount». Denne skulle brukes hver gang spillet starter hvor begge parter får to kort, men også i hit eller stand-situasjonene, hvor enten dealeren eller spilleren skal få ett kort av gangen.

Etter å ha delt ut kort til begge partene skulle spilleren få vite kortene sine og håndverdien sin. For ryddighetens skyld lagde jeg derfor en funksjon som skrev ut alle kortene med en liste som parameter. Brukeren skulle også få vite det første kortet til dealeren og verdien til dette kortet. Her hentet jeg bare ut det første kortet og verdien ved hjelp av indeksering.

Hoveddel:

Videre i oppgaven begynte jeg å forstå mer og mer selv, uten å måtte se på fremgangsmåten. Hittil har både brukeren og dealeren fått utdelt 2 kort hver og brukeren har fått sett kortene sine og dealerens første kort. Det neste jeg la til nå var funksjonen «check_for_blackjack». Det er egentlig her selve spille starter. Hvis brukeren fikk blackjack ville spillet vært over og if-testen vil returnere «blackjack», hvis ikke, vil funksjonen returnere funksjonen «userChoice».

UserChoice var det neste jeg ordnet. Dette var en funksjon som skulle gjentas flere ganger og var funksjonen som lot brukeren velge «hit» eller «stand», hvor inputet fra brukeren enten måtte være «h» eller «s» for å være gyldig. Hvis brukeren skrev «h» ville if-testen returnere en funksjon kalt «hit». Skrev brukeren inn «s» ville if-testen returnere funksjonen «stand».

Siden UserChoice returnerte stand eller hit, ble det å kode disse videre. Jeg begynte med «hit». Funksjonen starter med å gi spilleren ett kort og printer deretter ut dette kortet sammen med resten av kortene, før totalverdien av hånden blir skrevet ut. Videre sjekker funksjonen om du har fått en håndverdi over 21 som dermed betyr at spilleren har tapt og den sjekker om brukeren har fått blackjack. Hvis ingen av disse slår ut blir brukeren sendt til UserChoice igjen, hvor brukeren får velge om hen vil spille videre eller la dealeren spille.

Hvis brukeren velger stand på UserChoice er det dealeren sin tur til å spille. Her har jeg laget en while-løkke som går imens dealerens hånd er mindre enn spillerens hånd. I denne while-løkken får dealeren et nytt kort til løkken stopper. Dealeren spiller dermed helt til det er uavgjort, til dealeren har vunnet, eller dealeren har tapt. Etter dealeren har spilt ferdig, blir kortene printet ut, og det blir sjekket om dealeren vant, tapte eller gjorde det uavgjort

Avslutning:

Hittil har det ikke vært noen form for chips i spillet. Jeg lagde derfor en funksjon som spør brukeren hvor mange chips hen vil vedde. Før funksjonen blir definert, definerte jeg «player_chip = 5». Denne er utenfor selve spillet, slik at den ikke blir resatt som gjør det til en variabel som kan være gyldig over flere runder. I funksjonen er det en while-løkke som først spør om brukerinput. Deretter sjekkes det om input er gyldig på flere måter. Dette er da i en try-except-blokk med if-tester innenfor.

Den andre funksjonen jeg lagde i forbindelsen med chip håndtering var, «update_chips». Her er det en rekke if-tester som går gjennom resultatet av blackjack spillet. If testene sjekker om resultatet av spillet ble «blackjack», «win» eller «lose» og returnerer antall chips deretter.

Den siste funksjonen i spillet er «retry». Denne spør brukeren om hen vil prøve igjen. Svarer brukeren «y» for yes, sjekkes det om brukeren har nok chips til å starte en ny

runde. Har brukeren nok chips vil hen få starte på nytt, hvis ikke blir spillet avsluttet. Skriver brukeren «n» for no, avsluttes naturligvis også spillet.

Forbedringer:

Jeg er den typen som prøver litt fram og tilbake med løsninger før jeg bestemmer meg for hva jeg skal bruke. Grunnet dette var det ganske lite jeg ønsket og følte jeg trengte å forbedre, når det kommer til kodestruktur.

Det jeg derimot gjorde mot slutten av oppgaven var å legge inn dokumenterende, eller hjelpende kommentarer til hver funksjon, slik vi ble vist i forelesning. Jeg lagde også en klasse hvor jeg fant noen tagger for fet skrift og for understrek. Disse implementerte jeg slik at konsollen skulle være litt mer lesbar. Alle «input» fikk en understrek, slik at man lett kunne se hva som var bruker-input, og når brukeren valgte å hitte ble antall chips markert med fet tekst, slik at man tydelig ser hvor mange chips man har etter hvert hit.

Forbedringer som jeg tenkte på, men ikke gjennomførte er to stykker. Jeg startet hele oppgaven og valgte å bruke funksjoner. Noe jeg senere tenkte over, var at jeg kunne ha opprettet en klasse med metoder, istedenfor frittstående funksjoner. Det endte med å gå i glemmeboka hver gang jeg startet en «programmeringsøkt» og jeg fortsatte med funksjoner hver gang istedenfor.

Den andre forbedringen som jeg ikke gjorde, var å legge alle funksjonene i et eget dokument. Jeg tenkte at alle funksjonene kunne blitt lagt i et eget dokument, for ryddighetens skyld. Det endte med at jeg prøvde, men syntes at det ble for mye styr, for lite resultat. Selve «spillet» er på omkring 35 linjer hvor strukturen ikke hadde merket noe særlig forskjell. Jeg har også markert hvor «Main» starter, og så derfor ikke vitsen med å gjøre det. Hadde «Main» derimot vært lenger ville det heller vært mer hensiktsmessig å dele spillet og funksjoner.