Python Cheat Sheet

Variabler og Datatyper

Variabler

```
navn = "Ola"
alder = 30
```

Datatyper

```
tekst = "Dette er en tekst" # string
heltall = 42                # int
desimaltall = 3.14          # float
sannhet = True              # bool
```

Konvertering i mellom datatyper

```
s = "13"
print(type(s)) #
i = int(13)
```

Kontrollstrukturer

Betingelser

```
if alder > 18:
    print("Du er myndig")
else:
    print("Du er ikke myndig")
```

Løkker

Det finnes to typer løkker, `for`-løkker og `while`-løkker.

Løkker brukes til å gjenta noe flere ganger, feks

For å gjøre noe x antall ganger...

kan vi bruke for-løkker i kombinasjon med `range()`

```
for i in range(5): # skriver ut tallene fra 0 til 4
    print(i)
for i in range(1,6): # skriver ut tallene fra 1 til 5
    print(i)
```

Når man looper over elementer i en liste har ikke selve **variabelnavnet** mellom `for` og `in` noe å si for programmet:

```
frukter = ["eple", "banan", "appelsin"]
for frukt in frukter:
    print(frukt) # Skriver ut en frukt per linje
```

er det samme som

```
frukter = ["eple", "banan", "appelsin"]
for f in frukter:
    print(f) # Skriver ut en frukt per linje
```

> Den eneste forskjellen er **variabelnavnet** for hvert enkelt element som i eksempel #1 er `frukt` og i eksempel #2 er `f`

While-løkker

Vi kan bruke while-løkker til å gjøre det samme

```
i = 0
while i < 5: # skriver ut tallene fra 0 til 4
    print(i)
    i += 1 # inkrementerer i (gjør større)
```

```
i = 1
while i < 6: # skriver ut tallene fra 1 til 5
    print(i)
    i += 1 # inkrementerer i (gjør større)
```

```
frukter = ["eple", "banan", "appelsin"]
i = 0
while i < len(frukter): # skriver ut en frukt per linje
    print(frukter[i])
    i += 1
```

Timer: Eggkoker

1. Kalkuler antall sekunder
2. For hvert antall sekunder
    - Print gjenværende tid
    - Reduser antall sek. med 1
    - Vent (sleep) 1 sekund

```python
from time import sleep

sekunder = 60 * 7 # 7 minutter * 60 sek = 420 sek

while sekunder > 0:
    sekunder -= 1 # reduserer sekunder med 1
    print(f"{sekunder} sekunder igjen... ")
    sleep(1)
print("Egget ditt er ferdig!")
```

Funksjoner

ingen parametre

```
def si_hei():
    print("Hei")
si_hei() # Skriver ut: Hei
```

med parameter kalt `navn`

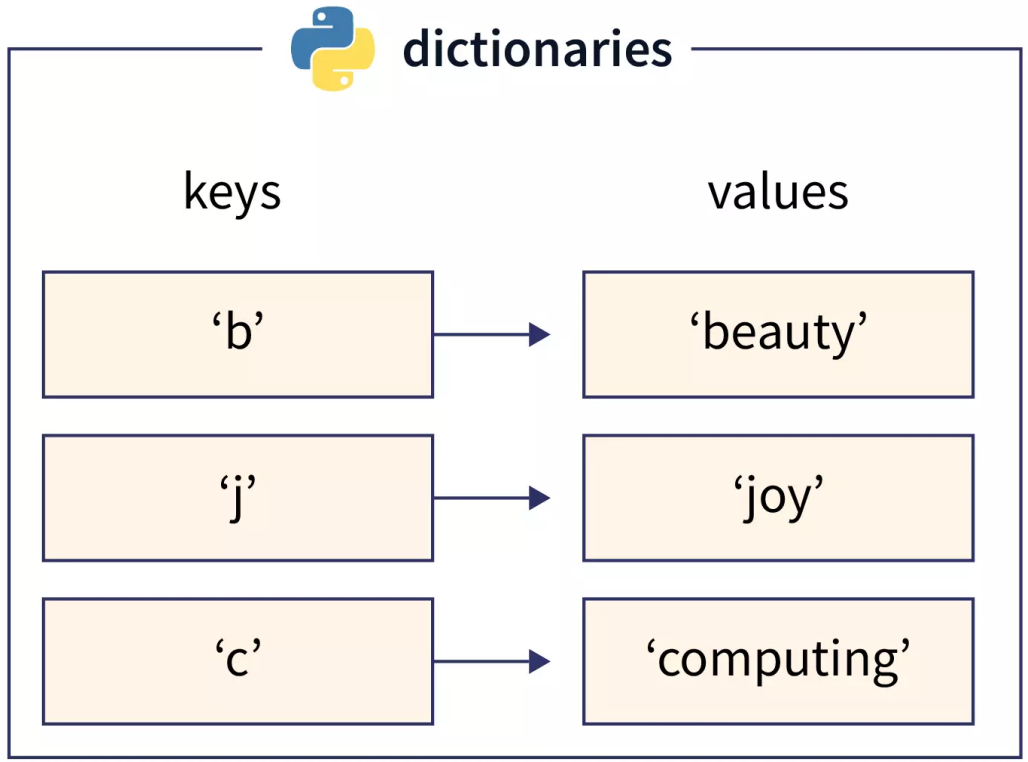
```
def hilsen(navn):
    print("Hei, " + navn + "!")
hilsen("Kari") # Skriver ut: Hei, Kari!
```

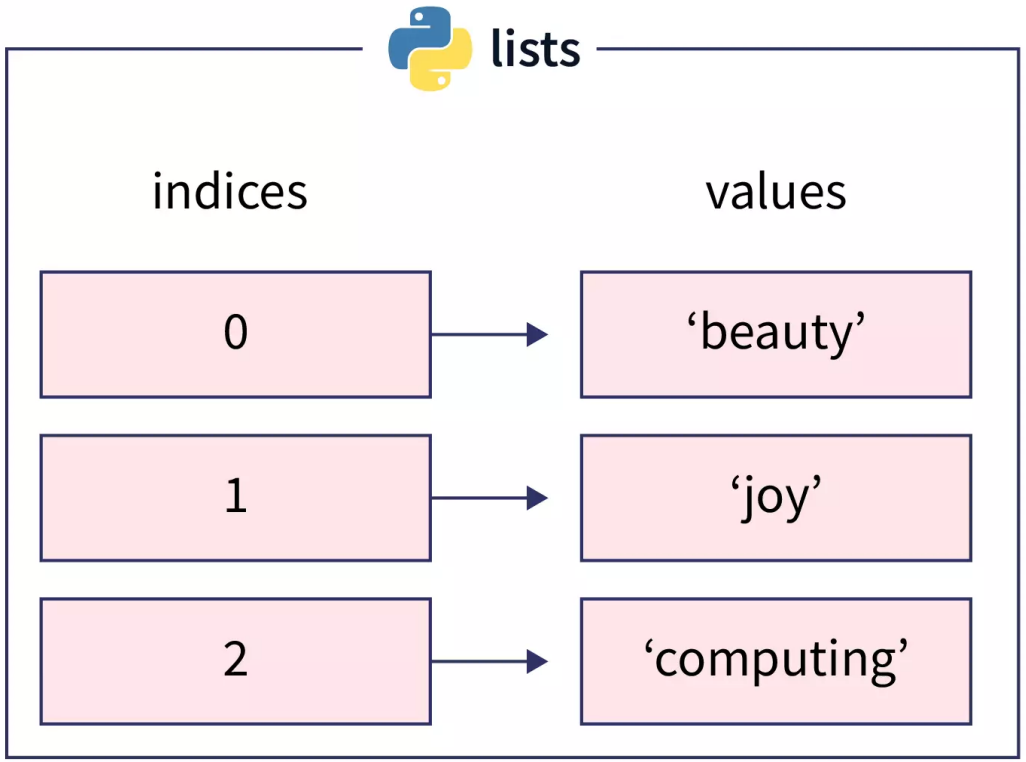
ingen parametre men returnerer en tekststreng

```
def si_hei():
    return "Hei"

svar = si_hei()
print(svar) # Skriver ut: Hei, Kari!
```

med parameter `navn` og returnerer en tekststreng

```
def hilsen(navn):
    return "Hei, " + navn + "!"

svar = hilsen("Kari")
print(svar) # Skriver ut: Hei, Kari!
```





SCALER Topics

Samlinger (collections) av data

Lister vs Dictionaries

- Både lister og dictionaries er samlinger av data, men har hver sitt bruksområde basert på hvordan du ønsker å organisere dataene.
- En liste i Python er en ordnet samling av elementer.
- En dictionary er en uordnet samling av nøkkel-verdi-par, der hvert element har en unik nøkkel for å muliggjøre rask tilgang til data.

Liste vs Dictionary eksempel

Dictionary

```
person = {"navn": "Per", "alder": 25}
print(person["navn"])  # Utskrift: Per
print(person["alder"])  # Utskrift: 25
```

Liste

```
frukter = ["eple", "banan", "appelsin"]
print(frukter[0])  # Utskrift: eple
print(frukter[1])  # Utskrift: banan
print(frukter[2])  # Utskrift: appelsin
```

Ofte brukte metoder på liste-objektet

- `append(obj)` legger til et nytt objekt i listen

```
frukter = ["eple", "banan", "appelsin"]
frukter.append("mango") # Legger mango til listen
print(frukter) # Utskrift: ['eple', 'banan', 'appelsin', 'mango']
```

- `extend(iterable)` legger til elementene fra et annet iterable (f.eks. liste eller tuple) til slutten av listen

```
frukter = ["eple", "banan", "appelsin"]
nye_frukter = ["kiwi", "jordbær"]
frukter.extend(nye_frukter)
print(frukter) # Utskrift: ['eple', 'banan', 'appelsin', 'kiwi', 'jordbær']
```

- `insert(index, obj)` setter inn et objekt på en spesifikk indeks i listen

```
frukter = ["eple", "banan", "appelsin"]
frukter.insert(1, "mango")
print(frukter) # Utskrift: ['eple', 'mango', 'banan', 'appelsin']
```

- `remove(obj)` fjerner det første forekomsten av et objekt i listen

```
frukter = ["eple", "banan", "appelsin"]
frukter.remove("banan")
print(frukter) # Utskrift: ['eple', 'appelsin']
```

- `pop(index)` fjerner og returnerer elementet på den angitte indeksen. Hvis indeksen ikke er spesifisert, fjernes og returneres det siste elementet.

```
frukter = ["eple", "banan", "appelsin"]
siste_frukt = frukter.pop()
print(siste_frukt) # Utskrift: appelsin
print(frukter) # Utskrift: ['eple', 'banan']
```

- `index(obj)` returnerer indeksen til det første forekomsten av et objekt i listen

```
frukter = ["eple", "banan", "appelsin"]
index_banan = frukter.index("banan")
print(index_banan) # Utskrift: 1
```

- `count(obj)` returnerer antall ganger et objekt forekommer i listen

```
tall = [1, 2, 3, 2, 4, 2, 5]
antall_toere = tall.count(2)
print(antall_toere) # Utskrift: 3
```

- `sort()` sorterer elementene i listen i stigende rekkefølge

```
tall = [5, 2, 8, 1, 3]
tall.sort()
print(tall) # Utskrift: [1, 2, 3, 5, 8]
```

- `reverse()` reverserer rekkefølgen på elementene i listen

```
frukter = ["eple", "banan", "appelsin"]
frukter.reverse()
print(frukter) # Utskrift: ['appelsin', 'banan', 'eple']
```

Dictionaries i Python

- **Definisjon:** Dictionaries er en uordnet samling av nøkkel-verdi-par i Python.
- **Bruksområde:** Egnet når du trenger å lagre data med tilknyttede nøkler for rask tilgang.
- **Eksempel:**

```
person = {"navn": "Per", "alder": 25}
print(person)  # Utskrift: {'navn': 'Per', 'alder': 25}
```

Ofte brukte metoder på dictionary-objektet

- `keys()` : Returnerer en liste med alle nøklene i dictionarien.

```
nøkler = person.keys()
print(nøkler)  # Utskrift: ['navn', 'alder']
```

- `values()` : Returnerer en liste med alle verdiene i dictionarien.

```
verdier = person.values()
print(verdier)  # Utskrift: ['Per', 25]
```

- `items()`: Returnerer en liste med nøkkel-verdi-par som tupler.

```
par = person.items()
print(par)  # Utskrift: [('navn', 'Per'), ('alder', 25)]
```

- `get(key)`: Returnerer verdien assosiert med den gitte nøkkelen, eller `None` hvis nøkkelen ikke finnes (kan også spesifisere en standardverdi).

```
alder = person.get("alder")
print(alder)  # Utskrift: 25
```

- `pop(key)`: Fjerner og returnerer verdien assosiert med den gitte nøkkelen.

```
navn = person.pop("navn")
print(navn)  # Utskrift: 'Per'
```

- `update(dictionary)`: Oppdaterer dictionarien med nøkkel-verdi-par fra en annen dictionary.

```
ny_info = {"yrke": "ingeniør"}
person.update(ny_info)
print(person)  # Utskrift: {'alder': 25, 'yrke': 'ingeniør'}
```

- `clear()`: Tømmer dictionarien for alle nøkler og verdier.

```
person.clear()
print(person)  # Utskrift: {}
```

- `in` (med nøkkel): Sjekker om en bestemt nøkkel finnes i dictionarien.

```
if "alder" in person:
    print("Alder eksisterer i dictionarien.")
```

Filbehandling

Lesing fra fil

```
with open("fil.txt", "r") as fil:
    innhold = fil.read()
    print(innhold)
```

Skriving til fil

```
with open("ny_fil.txt", "w") as fil:
    fil.write("Dette er en ny fil.")
```

Moduler og Biblioteker

```
import math

radius = 5
omkrets = 2 * math.pi * radius
print("Omkrets:", omkrets)
```

Forskjellige typer import

Eksempel: Vi ønsker å lage et program som simulerer et terningkast

```
import random # Vi importerer hele biblioteket i sitt eget namespace
terningkast = random.randint(1,6)
print(terningkast)
```

```
from random import randint # Vi importerer bare funksjonen vi trenger
terningkast = randint(1,6)
print(terningkast)
```

```
from random import * # Vi importerer fra biblioteket til vårt namespace
terningkast = randint(1,6)
print(terningkast)
```

Feilhåndtering

```
try:
    resultat = 10 / 0
except ZeroDivisionError:
    print("Kan ikke dele på null.")
```

Bruk av `try` og `except`

I Python kan du bruke `try` og `except` for å håndtere unntak (feil) i koden din. Dette er nyttig for å sikre at programmet ikke krasjer når det oppstår uventede situasjoner. Her er en enkel guide med eksempler.

Grunnleggende syntaks

```
try:
    # Kode som kan føre til unntak
    # ...
except ExceptionType as e:
    # Håndter unntaket her
    # e er en referanse til unntaksobjektet
    # ...
```

Eksempel 1: Enkel divisjon

```
try:
    resultat = 10 / 0
except ZeroDivisionError as e:
    print(f"Feil: {e}")
    # Håndterer delt på null-feilen
    resultat = "Udefinert"

print(f"Resultat: {resultat}")
```

Eksempel 2: Filbehandling

```
filnavn = "ikkeeksisterende.txt"

try:
    with open(filnavn, 'r') as fil:
        innhold = fil.read()
except FileNotFoundError as e:
    print(f"Feil: {e}")
    # Håndterer filen som ikke finnes
    innhold = None

print(f"Innhold: {innhold}")
```

Eksempel 3: Input-vasking

```
def les_tall():
    while True:
        try:
            tall = float(input("Skriv inn et tall: "))
            break  # Hopper ut av løkken hvis input er gyldig
        except ValueError:
            print("Feil input. Skriv inn et gyldig tall.")

    return tall

bruker_tall = les_tall()
print(f"Du skrev inn: {bruker_tall}")
```

I eksempel 3 bruker vi en funksjon `les_tall` som brukeren kan kalle for å få et gyldig tall. Hvis brukeren skriver inn noe som ikke kan konverteres til et tall, vil det fange opp unntaket og be brukeren om å prøve igjen.

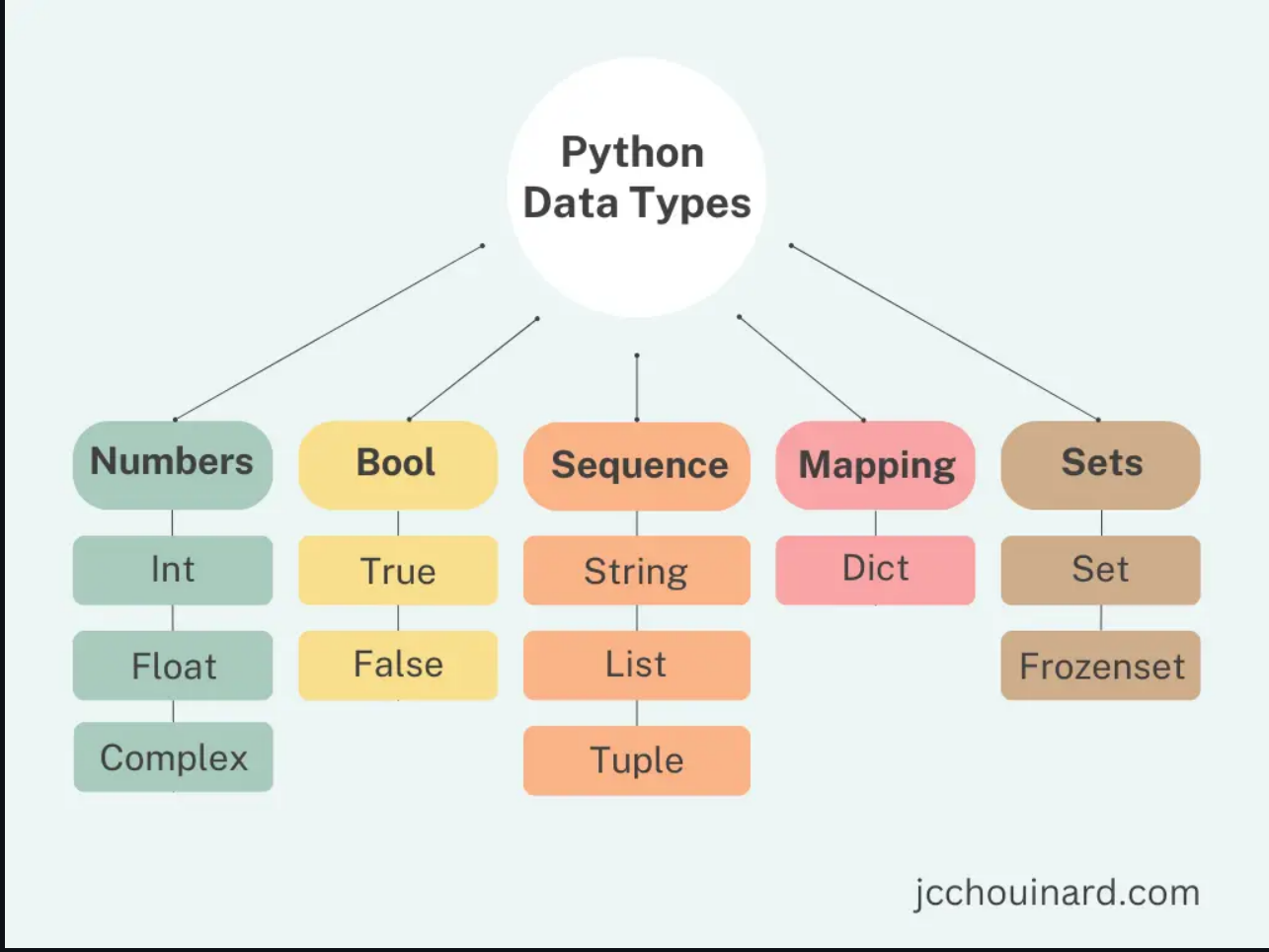
Dette er bare grunnleggende eksempler, men `try` og `except` er kraftige verktøy for å håndtere feilsituasjoner og gjøre koden din mer robust.

Oppgaver om types

Lag et program som demonstrerer bruken av de mest basic datatypene i python:

1. str
2. int
3. bool
4. float

Du skal lage et program som viser at du kan konvertere imellom de ulike datatypene, også vise at du kan handle errors når de oppstår.



Spørsmål

1. Hvilke ulike typer errors kan oppstå når man prøver å feks gjøre en str om til en int?
2. Hva menes med Duck Typing?

Python `type` Function Cheat Sheet

The `type` function in Python is used to determine the type of an object.

Syntax

```
type(object)
```

- `object` : The object whose type needs to be identified.

Example

```
x = 42
print(type(x))  # Output: <class 'int'>

y = "Hello, World!"
print(type(y))  # Output: <class 'str'>
```

Common Data Types

Numeric Types

```
a = 3.14
print(type(a))  # Output: <class 'float'>

b = 5
print(type(b))  # Output: <class 'int'>
```

Strings

```
text = "Python is awesome"
print(type(text))  # Output: <class 'str'>
```

Lists

```
my_list = [1, 2, 3]
print(type(my_list))  # Output: <class 'list'>
```

Tuples

```
my_tuple = (1, 2, 3)
print(type(my_tuple))  # Output: <class 'tuple'>
```

Dictionaries

```
my_dict = {'a': 1, 'b': 2}
print(type(my_dict))  # Output: <class 'dict'>
```

Booleans

```
flag = True
print(type(flag))  # Output: <class 'bool'>
```

Custom Classes

```
class MyClass:
    pass

obj = MyClass()
print(type(obj))  # Output: <class '__main__.MyClass'>
```

Remember that `type` returns the type as a class, and the class name is enclosed in single quotes. This cheat sheet covers some common use cases, but you can apply the `type` function to any Python object to determine its type.