Programmering 1 – Ruby Cheat Sheet v1.0 (rev 2019-05)

Köra ruby-program (i kommandotolken cmd.exe)

Ange kommandot **ruby** följt av namnet på filen som innehåller källkoden:

```
C:\>ruby my_source_code.rb
```

Navigera mellan mappar med kommandot **cd**:

```
C:\>cd MyFolder          byt mapp till MyFolder
C:\MyFolder\>cd ..       backa till föregående
```

Interactive Ruby Shell (irb)

Testa syntax och kodsnuttar med **irb**:

```
C:\>irb
irb(main):001:0>puts "Hello World!"
```

Värden, variabler och datatyper

Ett **värde** är ett stycke information (data) t ex talet 42, bokstaven K, ordet banan. En **variabel** är ett namn som symboliserar ett värde i koden.

Värden delas in i olika **datatyper** som har olika egenskaper och tillåtna operationer. Det går t ex inte (logiskt) att jämföra heltal med text på ett meningsfullt sätt.

Datatyp	**Benämning**	**Exempel**
Integer	heltal, int	843
String	sträng, text	"squiggle"
Boolean	bool	true false
Float	flyt- / decimaltal	34.8
Array	lista, fält	[4,7,1,9,12]

Input från användare

Funktionen **gets** (get string) låter användaren mata in en sträng som avslutas med enter. **chomp** tar bort radslutstecknet i slutet på strängen:

```
name = gets.chomp
```

Med **to_i** omvandlas input till ett heltalsvärde:

```
value = gets.to_i
```

Identering (formatera källkod)

Korrekt **indentering** (indrag av kod-block med tab eller blanksteg) gör källkoden mer lättläst och enklare att felsöka. VS Code och andra kod-editorer kan göra detta automatiskt.

Tilldelning (assignment), initiering

En variabel har ingen relevans i koden förrän den tilldelats ett värde. Efter **tilldelning** symboliserar variabeln sitt tilldelade värde:

```
name = "Berit"      # tilldelning
puts name           # name innehåller "Berit"
```

Den första tilldelningen av en variabel i koden kallas **initiering**. Vid nästa tilldelning av samma variabel kommer värdet att förändras (skrivas över). **Tilldelningsoperatorn =** ska inte förväxlas med jämförelseoperatorn **==**

Namngivning

Variabelnamn skall skrivas med **små bokstäver** (gemener). Inled inte med siffra. Undvik specialtecken (ääö, etc) och stora bokstäver (versaler) som ger konstant variabel. Använd **snake_case**, undvik CamelCase (konstant var). Reserverade ord som while, end, return osv kan inte användas som variabelnamn.

Tydlig namngiving innebär att variabler och funktioner ges *namn som syftar till deras innehåll eller funktion* i koden. Ex:

```
first_name = "Gunnar"          # bra namngiving
l33t_h4xx0r = "Gunnar"         # dålig
```

Kommentarer i källkod

Inleds med **#** och kan skrivas *efter en kodrad*:

```
sum = a + b       # summera talen a och b
```

Om raden *inleds med* # är hela raden kommentar:

```
# sum = a + b     (den här koden utvärderas ej)
```

Utskrift på skärmen (console output)

```
puts "Hello"   # utskrift med radbrytning
print "Hello"  # utskrift utan radbrytning
```

Funktionen **p** skriver ut på ett format som visar utskriftens datatyp (bra vid debugging!):

```
p "Hello"
```

Vid utskrift kan variabler integreras i en given sträng genom **sträng-interpolering**:

```
age = 18
puts "You are #{age} years old"
```

If-satser (if-statements)

if-satser har ett eller flera kod-block som utförs om ett **villkor** är sant. Endast ett block utförs (även om flera villkor är sanna). Ex:

if-elsif-else:

```
if x < y
  puts x
elsif x >= y
  puts y
else
  puts x + y
end
```

if-else:

```
if x > y
  puts x
else
  puts y
end
```

Operatorer för jämförelse (comparison)

Jämförelse måste ske mellan värden av samma datatyp. En jämförelse utvärderar alltid till **true** eller **false**.

```
x < y     x mindre än y
x > y     x större än y
x <= y    x mindre än eller lika med y
x >= y    x större än eller lika med y
x == y    x lika med y
x != y    x inte lika med y
```

Aritmetiska operatorer (numeriska värden)

```
a + b     addition
a – b     subtraktion
a * b     multiplikation
a / b     division
a % b     modulus (rest vid heltalsdivision)
a ** b    exponent ("a upphöjt i b")
```

Logiska operatorer

Används mest för att *kombinera flera villkor*. Använd gärna parenteser för komplexa villkor.

```
&&      logiskt "och", and fungerar likadant
||      logiskt "eller", or fungerar likadant
!       logiskt "inte", not fungerar likadant
```

Typomvandling (type conversion)

Omvandla värden till andra datatyper:

```
25.to_s      # to string -> "25"
"42".to_i    # to integer -> 42
"1.23".to_f  # to float -> 1.23
```

Programmering 1 – Ruby Cheat Sheet v1.0 (rev 2019-05)

Iteration med while-loop

En **while-loop** består av ett **villkor** och ett kod-block som upprepas (itereras) så länge villkoret är sant. Ex:

```
i = 0          # loop-varibel initieras
while i < 5    # kod-blocket upprepas fem ggr
  puts i
  i += 1       # inkrementera loop-varibel
end
```

I en **inkrementerande loop** (exemplet ovan) är *villkoret beroende av ett ökande värde*. I en **dekrementerande loop** är *villkoret beroende av ett minskande värde*, ex:

```
i = 5          # loop-variabel initieras
while i > 0    # kod-blocket upprepas fem ggr
  puts i
  i -= 1       # dekrementera loop-variabel
end
```

Nyckelordet **break** avbryter loopen direkt.
Nyckelordet **next** påbörjar nästa iteration/varv.

Indexering

Enskilda tecken eller värden som ingår i en sträng eller lista/array kallas **element**. Varje element har ett unikt **index** (numrerad position) som ger åtkomst till elementet. *Index är noll-baserat* och ökar när man går åt höger. Ex:

element (tecken):	B	e	r	t	i	l
index:	0	1	2	3	4	5

Indexerings-operatorn [] skrivs efter variabeln eller värdet som indexeras. Index anges inom klamrarna som värde eller variabel. Ex:

```
name = "Bertil"
puts name[4]   # "i" skrivs ut
name[0] = "b"  # ersätt element -> "bertil"
```

Speciella index-format:

```
name[-1]       # ger sista elementet -> "l"
name[-2]       # ger nästa sista osv -> "i"
name[1..3]     # intervall av element -> "ert"
```

Flödesschema (flowchart)

En grafisk representation av **programflödet** med symboler för **sekvens**, **selektion** (if-sats) och **iteration** (loop). Dessutom används särskilda symboler för **input** och **output**.

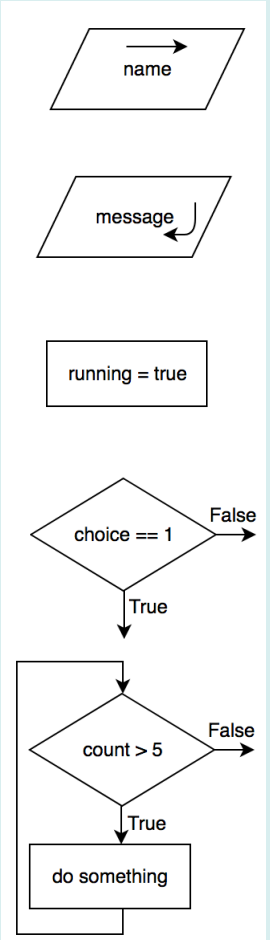
Input – en snedställd rektangel med en höger-riktad pil. Variabeln som tilldelas anges i symbolen.

Output – en snedställd rektangel med en "retur"-pil. Variabeln eller värdet som matas ut anges i symbolen.

Sekvenssteg eller ovillkorligt steg – en vanlig rektangel. Operationen som utförs skrivs som kod eller pseudokod.

If-sats – en "diamant" med ett villkor i kod eller pseudokod. Symbolen har *vardera en utgång för true och false*.

Loop – en diamant med ett villkor där *flödet återvänder till villkoret* om det är sant. När villkoret inte uppfylls går flödet vidare.



Sträng-operationer (ett litet urval)

```
"Hello\n".chomp    # tar bort radslut -> "Hello"
"Hello".length     # ger längd -> 5
" Hello ".strip    # tar bort space -> "Hello"
"one two".split    # ger lista -> ["one","two"]
"Hello".upcase     # versaler -> “HELLO"
"Hello".downcase   # gemener -> “hello"
```

Egendefinerade funktioner (metoder)

En funktion kapslar in ett stycke kod med en tydlig och avgränsad uppgift. Input till en funktion kallas **argument** eller **parametrar** (noll eller flera). Output kallas **returvärde**. Funktionen måste vara *definierad innan den anropas*. Variabler inom funktionen är lokala (inte tillgängliga utanför funktionskroppen).

```
def sum_equal( a, b )  # a och b är argument
  sum = a + b
  if sum % 2 == 0
    return true        # returvärde
  else
    return false       # returvärde
end                    # slut funktionskropp

sum_equal( 2, 4 )      # funktionsanrop
```

Array (lista, fält)

En lista är en **datastruktur** som innehåller **element** av en viss datatyp, t ex heltalsvärden eller strängar. Listor har *många gemensamma egenskaper med strängar*, t ex **indexering**.

```
my_list = []         # initiera tom lista
# skapa lista av strängar:
numbers = ["one","two","three"]
numbers[1]           # indexering -> "two"
```

Lägg till element i slutet av listan genom att inkrementera listans högsta giltiga index:

```
numbers[3] = "four" # lägg till fjärde element
```

För att **ta bort element** kan man skapa en ny lista och kopiera över valda element, ex:

```
new_list = numbers[0..2]
```

Några begrepp

Kontrollstruktur – if-sats eller loop
Argument – input till funktioner och program
Input-loop – styrs av användarens input
Inkrementera – öka värde (vanligtvis med ett)
Dekrementera – minska värde
Syntaxfel – uppstår när språkets regler inte följs, t ex parentesfel eller "end" som saknas.
Logiska fel – koden går att köra men ger felaktiga resultat, t ex == förväxlas med =
Algoritm – recept eller standardlösning på ett givet problem.
Interaktivt program – interagerar med användaren genom meddelanden och input.