Tölvunarfræði - Fylgiskjal

Í þessu skjali er farið yfir helstu atriðin í Python (og forritun almennt) sem þið þurfið á að halda til að leysa verkefnin

Athugasemdir

Athugasemdir eru notaðar til þess að útskýra kóðann eða sem minnispunkta fyrir forritarann. Athugasemdir byrja á `#` og svo kemur athugasemdin t.d.

```
# þetta er athugasemd!
```

Breytur

Breytur eru notaðar til að geyma gögn og þær verða til um leið og við setjum þær samasem þá gagna týpu sem við viljum geyma. Í þessu dæmi er `nafn` breytan og `"Jane"` er það sem við viljum geyma í breytunni.

```
nafn = "Jane"
```

Í Python þurfum við ekki að skilgreyna týpuna sem við viljum geyma í breytunni. Heiti á breytum geta verið stuttar eins og til dæmis x eða y, eða lýsandi eins og nafn, aldur, meðaltal o.s.frv..

Breytu nöfn

Breytur verða að: - Byrja á bókstaf a-z eða undirstriki . - *Breytur mega bara innihalda stafi, tölustafi , undirstriki (A-z, 0-9, and*). - Breytur eru hástafanæmar sem þíðir að *nafn, Nafn, og NAFN* eru þrjár mismunandi breytur. - Breyta má **EKKI** byrja á tölustaf. - Breyta má ekki vera eitt af Python lykilorðum (keywords) → sjá lista aftast.

Það er oft erfitt að lesa breytur með fleiri en eitt orð en það er hægt að gera margt til að einfalda það. Við getur notað undirstrik `_fjöldi_sæta`. Við getum notað eitthvað sem heitir Camel Case 🐫 þar sem að hvert orð eftir fyrsta orðið byrjar á stórum staf `fjöldiSæta`. Við getur líka notað eittvað sem heitir snake case 🐍 en þá notum við undirstriki `fjöldi_sæta_í_sal`.

Inntak frá notanda

Til að biðja notanda um inntak, er hægt að nota `input()` og vista það í breytu.

```
print("Hvað heitir þú?")
nafn = input()
print("Góðan daginn " + nafn  + "!")
```

`

Print

Í Python notum við `print()` fallið til að skila því sem við erum að gera.

```
>>> x = 5
>>> print(x)

5
```

Við getum skilað mörgum breytum í `print()` fallið með því að aðskilja þær með kommu `,`.

```
>>>x = "Python "
>>>y = "er "
>>>z = "skemtilegt "

>>>print(x, y, z)
Python er skemmtilegt
```

Við getum merkt staði í streng sem skipta á út fyrir með slaufusvigum `{}`. Passa þarf að viðföngin komi fyrir í réttri röð.

```
>>>x = "0"
>>>y = "32"

>>>print("{}°C eru {}°F", x, y)
0°C eru 32°F
>>>print("{}°C eru {}°F", y, x)
32°C eru 0°F
```

Gerðir gagna

Hægt er að sjá af hvaða gerð gögnin eru sem við erum að vinna með með því að nota `type()` fallið.

```
>>>x = 5
>>>print(type(x))
<class 'int'>
```

Hérna eru nokkrar algengar gerðir gagna:

Syntax	Type
x = “Hello World”	str
x = 20	int
x = 20.5	float

Tölur

Í Python eru tvær gerðir af tölum:

- `int`
- `float`

Breytur meða tölu gagna gerðini eru búnar til þegar meður setur breytu samasem tölu.

```
x = 1    # int
y = 2.8  # float
```

Eins og var útskýrt hér á undan þá getum við notað `type()` til að sjá af hvaða tölu gerð gögnin eru.

Int

Int eru jákvæðar eða neikvæðar heiltölur án kommu.

```
x = 1
y = 35656222554887711
z = -3255522
```

Float

Float eða (floating point number) eru jákvæðar eða neikvæðar kommu tölur með einn eða fleiri aukastafi.

```
x = 1.10
y = 1.0
z = -35.59
```

Strengir

Strengir eru umluktir gæsalöppum sem eru annað hvort einfaldar `'` eða tvöfaldar `"`. Við getum gert strengi sem eru nokkrar línur með því að nota þrjár tvöfaldar gæsalappir `"""` eða þrjár einfaldar gæsalappir `'''`. Við getum sé hversu langur strengurinn er mað því að nota `len()` fallið.

```
>>>a = "Hello, World!"
>>>print(len(a))

11
```

`

`

Samskeyting

Í forritun virkar plúsin eins og í stærðfræði hann leggur saman tvær tölur. En eð við notum + með strengjum þá skeytir hann strengina saman.

```
>>>fornafn = "Viktor"
>>>eftirnafn = "Hollanders"

>>>print(fornafn + eftirnafn)
ViktorHollanders
```

Eins og þið sjáið þá varð þetta eitt orð þrátt fyrir að við settum bil á milli `fornafn`, `+`, og `eftirnafn` . Þetta er vegna þess að samskeytingin les ekki bilin. Við þurfum sjálf að setja þau inn á milli gæsalappana. Við getum gert að með því að bæta inn streng sem er bil `" "` eða með því að bæta bil fyrir aftan fornafn `"Viktor "` eða fyrir framan eftirnafn `" Hollanders"`. Hérna er þatta sýnt með bill inn í streng

```
>>>print(fornafn + " " + eftirnafn)
Viktor Hollanders
```

Samskeyting á mismunandi týpum

Athugið að **ekki** er hægt að skeyta saman tveimur mismunandi týpum, t.d. int og streng.

```
>>> nafn = "Paul"
>>> aldur = 16
>>> print (nafn + " er " + aldur + " ára")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Í þessu tilfelli þurfum við að að nota `str()` fallið sem breytir int í streng

```
>>> print (nafn + " er " + str(aldur) + " ára")
Paul er 16 ára
```

Samskonar fall `int()` er í boði ef breyta skal streng yfir í int.

```
>>> aldur = "16"
>>> aldurEftirTvoAr = aldur + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> aldurEftirTvoAr = int(aldur) + 2
>>> print(aldurEftirTvoAr)
18
```

En passa þarf sérstaklega upp á að inntakið innihaldi tölustafi eins og fjallað verður um í kaflanum **Villumeðhöndlun**.

Villumeðhöndlun

Upp geta komið ýmisskonar villur þegar beðið er um inntak frá notanda. Til að bregðast við því er í boði að nota `try`-segðina. Allur kóði sem er keyrður innan um `try` er hægt að "grípa" með `except` og bregðast þannig við. Dæmi má sjá í næsta undirkafla.

Nota int() á bókstafi

Passa þarf upp á að **strengurinn sé örugglega tölustafur** þegar `int()` er notað.

```
>>> aldurBókstafir = "sextán"
>>> aldurTölustafir = int(aldurBókstafir)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'bla'
```

Oft fær forritið inntak frá notanda og því ekki hægt að vera viss um að það sem slegið er inn sé það sem beðið er um. Þá er hægt að "grípa" villur sem gætu átt sér stað og annaðhvort beðið notandann um að reyna aftur, eða slökkva á keyrslunni "gracefully".

```
print ("Sláðu inn aldur: ")
aldurInntak = input()
aldur = None
while (aldur is None):
 try:
     aldur = int(aldurInntak)
 except ValueError:
     print("Inntak verður að vera tölustafur")
```

Takið eftir að forritið hér fyrir ofan keyrir þangað til að tölustafur er sleginn inn.

Aukaverkefni: Bætið við forritið þannig að aðeins tölustafir á milli 0 og 100 eru leyfilegir.

Slökkva á forriti "gracefully"

Stundum kemur upp villa í keyrslunni á forriti að ekkert annað er í stöðunni en að stoppa. Þetta getur gerst þegar notandinn slær inn viðfang sem forritið getur ekki unnið með. Til þess að stoppa forritið í miðri keyrslu, þarf að nota `import` skipunina, sem flytur inn föll annarsstaðar frá, sem forritarinn getur notað án þess að forrita þau upp á eigin spýtur.

Venjulega er `import` skipunin sett í fyrstu línu forritsins, og í sumum forritunarmálum er það algjört skilyrði. En í Python má hún kom fyrir hvar sem er í forritinu.

```
if (tilraunir == 0):
    print("Game over, bæ bæ")
    import sys
    sys.exit(0)
```

Escape Characters

Eins og þið hafið tekið eftir þá notum við suma stafi sem við myndum kanski vilja nota inn í strengi til dæmis gæsalappir. Við getum ekki bara gert tvöfaldar gæsalappir inn í streng sem er þegar með tvöfaldar gæsa lappir því þá fáum við villu. `txt = "We are the so-called "Vikings" from the north."` Þetta orsakar villu. Í staðinn þurfum við að nota Escape Character sem er öfugt skástrik og svo stafurinn sem við viljum nota `\"`.

```
txt = "We are the so-called \"Vikings\" from the north."
```

Boolean

Boolean getur annað hvort verið **True** eða **False.**

```
>>>x = 5
>>>booleanDaemi = x > 4
>>>print(booleanDaemi)
True
```

Auk þess er hægt að beita aðgerðum, eða virkjum, á Boolean-gildi. En ólíkt tölum sem hægt er að leggja saman, draga frá, margfalda eða deila, þá standa öðruvísi aðgerðir til boða. Dæmi eru `and` og `or` hér fyrir neðan.

```
>>>booleanDaemi2 = x < 0
>>>print(booleanDaemi and booleanDaemi2)
False
>>>print(booleanDaemi or booleanDaemi2)
True
```

Seinna dæmið er `False` því **báðar** breytur verða að vera True. Ef aðeins önnur þeirra þarf að vera True, skal nota `or`.

Hinsvegar er hægt að nota neitunarvirkja, `not` eða `!` til að “flippa” Boolean gildi.

```
>>>print(booleanDaemi and not booleanDaemi2)
True
```

Þar sem `booleanDaemi2 == False`, þá verður `not booleanDaemi2 == True`, sem er rökrétt.

Meira um það í næsta kafla.

Aðgerðir

Python skiptir aðgerðum í eftirfarandi flokka:

- Reikniaðgerðir
- Gildingaraðgerðir
- Röksamanburðaraðgerðir
- Röklegar aðgerðir

Reiknisaðgerðir

Reikniaðgerðir kannast flestir við en það eru:

Aðgerð	Heiti	Dæmi
+	Samlagning	x + y
-	Frádráttur	x - y
*	Margföldun	x * y
/	Deiling	x / y
%	Eftirstöðvun	x % y
**	Veldissetning	x ** y

Gildisveitingaraðgerðir

Gildisaðgerðir eru aðgerðir sem setja eitthvað gildi jafnt og eitthvað. Í bland við reikniaðgerðir er hægt að nota gildisaðgerðir sem styttingu, þannig að í stað þess að gera `x = x + 3` þá getum við gert `x += 3`. Þetta er hægt að gera með allar reiknisaðgerðirnar.

Aðgerð	Stytting dæmi	Dæmi
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
	=	x

Eftirfarandi rökaðgerðir bera saman tvö gildi.

Tákn	Heiti	Dæmi	Þýðing
`<`	Minna en	`x < y`	x er minna en y
`>`	Stærra en	`x > y`	x er stærra en y
`==`	Jafnt og	`x == y`	x er jafnt og y
`not` eða `!`	Neitun	`not x`	x er ekki satt
`is not` eða `!=`	Neitun	`x != y`	x er ekki jafnt og y

Þessar aðgerðir eru mikið notaðar í `if` og `while`

```
if (x != y):
    #kóði hér
...
while (x > y):
    #kóði hér
```

Röklegar aðgerðir

Röklegar aðgerðir eru notuð til að sameina skilyrði

Aðgerð	Heiti	Dæmi
`and`	Skilar `True` ef báðar fullyrðingarnar eru sannar	`x < 5 and  x < 10`
`or`	Skilar `True` ef önnur fullyrðingin eru sönn	`x < 5 or x < 4`
`not`	Skilar `False` ef útkoman er `True`, snýr niðurstöðuni við	`not(x < 5 and x < 10)`

None

Stundum viljum við taka frá breytu en ekki gefa því neitt gildi. Til þess er hægt að nota `None`.

```
a = None
while (a is None):
    b = input() #inntak frá notanda
    if (b is "c" or b is "f"):
        a = "ekki lengur None"
#Kóði hér með rétt inntak frá notanda
```

Í dæminu fyrir ofan keyrir forritið í lykkjunni þar til inntak frá notanda er eins og við má búast.

Segðir í Python

If-segðin

Hægt er að keyra kóða undir ákveðnum skilyrðum. Þetta er gert með `if`-segðinni sem tekur inn rökaðgerð og tölvan metur hvort kóðinn sé keyrður í framhaldinu.

Einnig er í boði að bæta við `elif`, sem er stytting á `else if`. Þessi kóði keyrir aðeins ef skilyrðið í `if`-segðinni á undan var ekki uppfyllt. Að lokum er í boði `else`, en það kemur aldrei skilyrði á eftir henni, m.ö.o. það fylgja ekki svigar á eftir. Þessi kóði er keyrður ef engin skilyrði á undan urðu uppfyllt.

```
if (unit == "c"):
    print("Talan er í celsíus")
elif (unit is "f"):
    print("Talan er í Fahrenheit")
else:
    print("Villa")
```

Takið eftir að eftir skilyrðinu í fyrstu línunni kemur tvípunktur : Þær línur sem koma á eftir tvípunktinum verða að byrja með **tab**-bili, annars tekur tölvan ekki mark á skilyrðinu.

Ýmsir möguleikar til að bera saman tvö gildi eru í boði, í forritinu fyrir ofan er athugað hvort að breyta sé jöfn “c” eða “f”. Fleiri samanburðaraðgerðir eru að finna í kaflanum **Aðgerðir** hér fyrir neðan.

Takið einnig eftir að hægt er að nota bæði `is` eða `==`, en það virkar alveg eins.

Lykkjur

Lykkjur keyra sama kóða aftur og aftur þar til skilyrði er mætt. Til eru tvær helstu gerðir af lykkjum, `while` og `for`.

```
i = 0
while (i < 10):
   print("Þessi lína verður prentuð 10 sinnum")
   i = i + 1 #hér má líka skrifa i += 1 sem gerir nákvæmlega það sama.
```

for lykkjur eru gjarnan notaðar til að ítra yfir gagnasöfn, t.d. fylki eða skrá.

```
file = open("vedurgogn.txt", "r") #hér stendur r fyrir read og gefur tölvunni
         til kynna að skráin verður lesin.
for (line in file):
    print (line)
```

Í dæminu fyrir ofan er skrá opnuð með lesaðgang og prentuð út á skipanagluggann.

Fylki

Fylki (e. array) virka eins og listar, þar sem mörg gildi eru vistuð undir sömu breytunni. Fylki eru táknaðar með hornklofum `[]`

```
cars = ["Ford", "Volvo", "BMW"]
```

Til að sækja gildi úr fylki, er aftur notaðir hornklofar með tölustaf sem vísar í hvaða sæti gildið er.

ATH í Python og flestum forritunarmálum er fyrsta gildið alltaf 0.

```
>>> print(cars[0])
Ford
>>> print(cars[1])
Volvo
>>> print(cars[2])
BMW
```

Stærð fylkis

Hægt er að komast að stærð fylkis með fallinu `len()`.

```
>>>print(len(cars))
3
```

Hafa þarf aðgát þegar `len()` er notað til að sækja gögn úr fylki, þar sem stærðin á fylkinu mun valda villu.

```
>>> cars[len(cars)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Þetta kemur til því að fyrsta gildið er ávallt 0, þess vegna er síðasta gildið í fylkinu `len(cars) - 1`, eða í þessu tilfelli 2, eins og sést fyrir ofan.

Til að sækja síðasta gildi fylkis, þarf að draga frá 1.

```
>>> cars[len(cars) - 1]
BMW
```

Ítrun yfir fylki

Til að spara okkur sporin er hægt að nota lykkjur. ### while Með `while`-lykkju þarf að finna út stærðina á fylkinu, en það er gert með áðurnefnda `len()` fallinu.

```
>>>i = 0
>>>while (i < len(cars)):
>>>     print (cars[i])
>>>     i += 1
Ford
Volvo
BMW
```

for

Einnig er hægt að nota `for`-lykkju. Takið eftir að hér þarf ekki að nota hornklofa eins og í `while`-lykkjunni og kóðinn er snyrtilegri fyrir vikið.

```
>>>for (car in cars):
>>>    print (car)
Ford
Volvo
BMW
```

Föll

Fall er kóða blokk sem keyrir ákveðinn kóða þegar við köllum á það. Þetta er kjörin leið til að endurskrifa ekki sama kóða.

Fall getur tekið inn viðfang og það getur líka skilað gögnum. Í Python er fall skilgreint með `def` lykilorðinu.

```
def fallið_mitt():
  print("Hæ frá falli")
```

Að kalla á Fall

Til þess að kalla á fallið notum við fallnafnið með sviga fyrir aftan

```
>>>def hallo():
>>>  print("Halló heimur")

>>>hallo()
Halló heimur
```

Viðföng

Við getum látið fallið hafa upplýsingar með því að setja inn sem viðfang. Viðfangið er sett inn í svigana.

```
>>>def heilsaðu(fnafn):
>>>  print("Hæ {}".format(fnafn))

>>>heilsaðu("R2D2")
Hæ R2D2
>>>heilsaðu("Yoda")
Hæ Yoda
>>>heilsaðu("Chewbacca")
Hæ Chewbacca
```

Það er hægt að setja inn mörg viðföng en maður þarf þá að aðskilja þau með kommu `,`.

```
>>>def undurVeraldar(bygging, byggt):
>>>  print("{}, byggingu lokið árið {}".format(bygging, byggt))

>>>undurVeraldar("Eiffel turninn", "1889")
Eiffel turninn, byggingu lokið árið 1889
>>>undurVeraldar("Pýramídarnir","2610 BC" )
Pýramídarnir, byggingu lokið árið 2610 BC
>>>undurVeraldar("Kína Múrinn" "206 BC")
Kína Múrinn, byggingu lokið árið 206 BC
```

Fall verður að vera kallað með réttan fjölda af viðföngum. Ef fallið tekur inn tvö viðföng, þá þurfum við að setja inn tvö viðföng. Ein undantekning eru sjálfgefin viðföng.

Sjálfgefin viðföng

Stundum viljum við hafa sjálfgefið viðfang ef ekkert viðfang er sett inn í fallið.

```
>>>def heimaland(land = "Íslandi"):
>>>  print("Ég er frá " + land)

>>>heimaland("Svíþjóð")
Ég er frá Svíþjóð
>>>heimaland("Frakklandi")
Ég er frá Frakklandi
>>>heimaland()
Ég er frá Íslandi
>>>heimaland("Bandaríkjunum")
Ég er frá Bandaríkjunum
```

Að skila gildum, return

Við getum notað `return` til að láta fallið skila gildi.

```
>>>def margfaldaðMeðTveimur(x):
>>>  return 2 * x

>>>print(margfaldaðMeðTveimur(3))
6
>>>print(margfaldaðMeðTveimur(5))
10
>>>print(margfaldaðMeðTveimur(9))
18
```