



Inngangur að forritun í Python

Forritun fyrir byrjendur

Valborg Sturludóttir



Copyright © 2020 Valborg Sturludóttir

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, Sept 2020

Efnisyfirlit

I	Fyrsti hluti	
1	Inngangur	9
1.1	Tilgangur bókarinnar	9
1.2	Hvers vegna Python?	10
1.3	Uppsetning	10
1.4	Að keyra kóða	10
1.5	Málskipan	12
1.5.1	Uppsetning á kóða	12
1.5.2	Gagnatýpur og lykilorð	13
2	Tölur og breytur	15
2.1	Tölur - talnatýpur	15
2.2	Reikniaðgerðir og tákn	16
2.3	Breytur	17
3	Strengir	21
3.1	Strengir skilgreindir	21
3.2	Strengir og reikniaðgerðir	22
4	Listar	25
4.1	Gagnagrindur	25

5	Segðir og skilyrðissetningar	27
5.1	Segðir	27
6	Lykkjur	29
6.1	while - á meðan eitthvað er satt	29
7	Orðabækur	31
7.1	Lyklar og gildi	31
8	Föll	33
8.1	Tilgangur falla	33
8.2	Að skrifa föll	33
8.3	Viðföng	33
8.4	Skilagildi	33
8.5	Lokun	33
8.5.1	Descriptions and Definitions	33
9	In-text Elements	35
9.1	Theorems	35
9.1.1	Several equations	35
9.1.2	Single Line	35
9.2	Definitions	35
9.3	Notations	36
9.4	Remarks	36
9.5	Corollaries	36
9.6	Propositions	36
9.6.1	Several equations	36
9.6.2	Single Line	36
9.7	Examples	36
9.7.1	Equation and Text	36
9.7.2	Paragraph of Text	37
9.8	Exercises	37
9.9	Problems	37
9.10	Vocabulary	37

II

Part Two

10	Presenting Information	41
10.1	Table	41
10.2	Figure	41

Bibliography	43
Articles	43
Books	43
Index	45

Fyrsti hluti

1	Inngangur	9
1.1	Tilgangur bókarinnar	
1.2	Hvers vegna Python?	
1.3	Uppsetning	
1.4	Að keyra kóða	
1.5	Málskipan	
2	Tölur og breytur	15
2.1	Tölur - talnatýpur	
2.2	Reikniaðgerðir og tákni	
2.3	Breytur	
3	Strengir	21
3.1	Strengir skilgreindir	
3.2	Strengir og reikniaðgerðir	
4	Listar	25
4.1	Gagnagrindur	
5	Segðir og skilyrðissetningar	27
5.1	Segðir	
6	Lykkjur	29
6.1	while - á meðan eitthvað er satt	
7	Orðabækur	31
7.1	Lyklar og gildi	
8	Föll	33
8.1	Tilgangur falla	
8.2	Að skrifa föll	
8.3	Viðföng	
8.4	Skilagildi	
8.5	Lokun	
9	In-text Elements	35
9.1	Theorems	
9.2	Definitions	
9.3	Notations	
9.4	Remarks	
9.5	Corollaries	
9.6	Propositions	
9.7	Examples	
9.8	Exercises	
9.9	Problems	
9.10	Vocabulary	



1. Inngangur

1.1 Tilgangur bókarinnar

Þessi bók fjallar um þau undirstöðu atriði sem þarf að kynna til að ná tökum á forritun í Python. Höfundur finnst mikilvægt að kenna námsefnið með íslenskum hugtökum þar sem ætlunin er að nota hana í kennslu í íslenskum framhaldsskólum. Ef nemendur ætla að leggja fyrir sig tölvunarfræði í framhaldssnámi er nauðsynlegt að búa yfir ríkulegu íðorðasafni, þess þá heldur ef nemandi hyggst framfleyta fræðunum. Hugtök verða þó líka sett fram á ensku því lesandi gæti óskað að fletta upp íterefni sem meira er til af á netinu á ensku en íslensku.

Það er algengur misskilningur að forritarar kunni rosalega mörg forritunarmál, eins og fólk sem getur talað mörg tungumál, eða að það að kunna rosalega mörg mál geri þig að góðum forritara. Þvert á móti. Að sýna hæfni og leikni í einu máli er auðveldlega yfirfæranlegt á önnur mál sé þess þörf. Þess vegna er spurningin „hvað kanntu mörg forritunarmál?“ út í hött. Ekki aðeins eru tungumál og forritunarmál gerólík, forritunarmál eru formleg mál og þekking á einu hlutbundnu máli er nær því að vera jafn frábrugðið öðru í grunninn eins og málýskur innan tungumála. Nær væri að spyrja hvort viðkomandi hafi meiri áhuga á framenda eða bakenda forritun, hvað er skemmtilegasta reikniritið sem viðkomandi hefur útfært eða hvert er það forritunarmál sem viðkomandi grípur oftast í.

Einnig er það algengur misskilningur að það fyrsta sem fólk gerir er að búa til tölvuleik. Það þarf mikla undirstöðu kunnáttu til þess að geta búið til tölvuleiki, alveg eins og áður en hafist er handa við að skrifa bók þarf að læra stafrófið. Þessi grunnvinna finnst mörgum vera leiðigjörn. Að mati höfundar er það vegna þess að við erum svo vön því að nota tölvur dagsdaglega, svo fræðigreinin sem tæknin byggir á hlýtur líka að vera okkur kunnug ekki satt? Nei, alveg eins og dýralækningar eru okkur ekki augljósar við að eiga gæludýr og pípulagnir heldur ekki við að eiga klósett. Innan tölvunar eru ákveðnar grunneiningar sem eru notandanum ekki augljósar, af góðri ástæðu, það væri hrikalegt ef við þyrftum öll að vera píparar til þess að geta notað klósett. Þó þessi samlíking hafi verið heldur gróf þá sýnir hún að það eru svo margir hlutar sem eru okkur huldír að við hreinlega vitum ekki hvað við vitum ekki. Því er nauðsynlegt að læra grunninn vel og fara rólega yfir hann svo þegar við ætlum að fara að afrita og líma kóða frá síðum eins og stackoverflow þá vitum við allavega hvað sá kóði gerir (nokkurn veginn).

Uppbyggingin er þannig að fyrri hlutinn snýr að því að kynna lesandann fyrir grunn virkni Python; málskipan, lykilhugtök og lykilorð, gagnatýpur, lykkjur og föll. Seinni hlutinn snýr svo að því að beita þekkingu úr fyrri hlutanum í hlutbundinni forritun. Þar eru kynntir til sögunnar klasar og aðferðir sem lesandinn útfærir upp á eigin spýtur. Ekki er búist við neinni fyrri kunnáttu við lestur þessarar bókar, hún á að geta staðið fyrir sínu án þess að lesandinn búi yfir nokkurri þekkingu á sviði tölvunarfræða eða forritunar. Ef slík þekking er fyrir hendi gæti lesandanum þótt ágætt að fara hratt í gegnum fyrri hluta bókarinnar og einbeita sér að verkefnum úr seinni hlutanum. Í gegnum bókina fylgjum við svo þremur verkefnum sem verða þyngri og flóknari eftir því sem fleiri hugtök eru kynnt til sögunnar.

1.2 Hvers vegna Python?

Ástæður þess að Python er gott mál til þess að byrja á að skoða eru eftirfarandi: ¹.

1. *Málskipanin* er mjög svipuð mannlegu máli svo það er auðvelt að læra hvernig eigi að „tala“ við tölvuna.
2. Python er *kvíkt tagað* forritunarmál, það þýðir að notandinn þarf ekki að gefa upp hvers konar *gagnatýpur* er unnið með. Þetta gerir það að verkum að notandinn þarf ekki að læra urmull af lykilorðum áður en byrjað er að forrita.
3. Python er ekki alveg *hlutbundið* forritunarmál, sem gerir það að verkum að notandinn þarf ekki að læra hvernig á að beita hlutbundinni forritun fyrr en góð undirstaða er þegar komin.
4. Python er frítt og aðgengilegt öllum helstu stýrikerfum og einnig er hægt að forrita yfir netið í vafra og því óþarfi fyrir notandann að setja nokkuð upp sé þess óskað.
5. Python er mikið notað, algengt mál svo það er praktískt að hafa undirstöðu skilning á því.
6. Nefnt í höfuð á Monty Python grínhópsins.

1.3 Uppsetning

Víðsvegar um bókina, aðallega í upphafi, má finna númeraða kóðabúta sem eru ekki teknir úr vinnubók og eru því ekki eins litakóðaðir og þeir sem eru teknir úr vinnubókum. Ástæðan fyrir því er að þessum kóðabútum er auðveldara að viðhalda heldur en skjáskotum úr vinnubókum og því er heldur vísað í bækur sem eru aðgengilegar lesendum og frumstæðari framsetning ræður heldur ríkjum hér.

1.4 Að keyra kóða

Það fyrsta sem nemendur vilja yfirleitt gera er að byrja að skrifa sinn eigin kóða. Áður en við komumst svo langt þarf að útskýra hvernig það er gert. Þessi kennslubók byggir á notkun Jupyter Notebooks með hjálp Anaconda hugbúnaðarins, sem er öflugt pakkakerfi og tólakista sem hefur upp á mikið meira en bara Jupyter að bjóða. Hægt er að nálgast Anaconda á anaconda.com. Hægt er að nota Jupyter án þess að ná í Anaconda með síðum eins og cocalc.com. Einnig er hægt að keyra kóða á netinu í gegnum síður eins og repl.it, nota ritla (eins og [notepad](https://notepad.com) eða [sublime](https://sublime.com)) til að keyra .py skrár í skipanalínu, eða nota þyngri umhverfi eins og [pycharm](https://pycharm.com) sem eru sérhönnuð fyrir hugbúnaðarþróun. Hér er gert ráð fyrir Jupyter umhverfinu og verður bókina öll miðuð að því.

Þessari bók fylgja einnig nokkrar vinnubækur úr Jupyter sem lesandinn getur nýtt sér. Hér á mynd sést hvernig tóm Jupyter vinnubók lítur út. Virkninni er skipt upp í sellur og keyrsluröð sellanna skiptir máli, við sjáum seinna mikilvægi þess að geta skipt upp kóða svona og hvers vegna

¹ Strax í þessum texta koma fyrir hugtök sem verða skýrð betur seinna, ekki missa kjarkinn.

Þetta umhverfi er þægilegt til að byrja í. En hver selli hefur aðgang að svokölluðu skilgreiningar-svæði vinnubókarinnar en er þó sín eigin eining, því má keyra eina sellu í einu án þess að keyra allan kóðann í vinnubókinni.

Hér væri réttast að skoða Vinnubók 1 sem fylgdi þessari bók.

En við keyrslu á kóða þarf einnig að hafa í huga að tölvan gerir nákvæmlega það sem við segjum henni að gera og ekki annað. Og þá komum við niður á stórt vandamál, að tölvur eru mjög bókstaflegar og vitlausar. Þær skortir allt vit, þær reyna ekki að hafa vit fyrir þér. Þær gera nákvæmlega það sem þú baðst um. Nákvæmlega eins og þú baðst um það.

Þannig að ef ég ætlaði að segja tölvu að smyrja handa mér hnetusmjörs og sultu samloku þá þyrfti ég að segja vélinni að gera eftirfarandi í nákvæmlega þessari röð:

1. taka fram hníf
2. taka fram tvær brauðsneiðar
3. opna hnetusmjörið
4. setja beitta endann ofan í hnetusmjörið þannig að hann nái upp 50gr af hnetusmjöri
5. setja hnetusmjörið sem er á hnífnum á miðja brauðsneiðina
6. nota hnífinn til þess að smyrja hnetusmjörinu á þá hlið sem hnetusmjörið er nú þegar á, og enga aðra
7. taka fram skeið
8. opna sultuna
9. setja kúpta enda skeiðarinnar ofan í sultukrukkuna
10. taka skeiðina upp úr sultukrukkunni með kúfaða skeið af sultu
11. setja sultuna á hina brauðsneiðina
12. nota skeiðina til að smyrja sultunni yfir þá hlið brauðsneiðarinnar sem sultan er á og enga aðra hlið
13. setja brauðsneiðarnar saman þannig að hnetusmjörið og sultan snertist og hornin mætast öll.

Takið eftir að hér er gert ráð fyrir þó nokkur og ef vélin kann ekki nú þegar skil á:

1. taka fram
2. hnífur
3. opna
4. mæla 50 gr
5. smyrja
6. hlið á brauðsneið
7. miðja á brauðsneið
8. skeið
9. kúfað

Svo þó svo að þér hafi þótt þessi útskýring á samlokugerð alveg ofboðslega óþarflega nákvæm þá er ekki víst að úr þessu verði nokkur samloka. Þetta könnumst við öll við, að tölvur gera það sem þeim er sagt, ekki það sem við viljum.

Helsta verkefni forritara er að búa niður verkefni í svo litla hluta að hægt er að útskýra þá fyrir tölvu. Ekki búast við því að setjast niður við fyrsta verkefni og ætlast svo til að búa til tölvuleik eða hakka banka. Forritun er einnig frábrugðin þeirri venjulegu tölvunotkun sem þú hefur vanist dagsdaglega. Þar ertu ekki að gefa tölvunni þínar eigin skipanir heldur ertu að beita skipunum sem aðrir forritara hafa samið og sett upp í hugbúnaðinn sem þú ert að nota.

hér væri gott að
fyrir dæmi úr vi
sem á að fylgja

segja miklu meir
um hvernig á að
vinnubækur yfir

1.5 Málskipan

Málskipan (e. syntax) er hugtak sem þýðir hvernig á að skrifa kóða svo að hann þýðist í vélamál sem tölvur skilur. Málskipan eru þær reglur sem við þurfum að fara eftir þegar við forritum, þær reglur sem forritunarmálið býst við að við förum eftir. Ef við brjótum þessar reglur fáum við villu, og einhver algengasta villa sem hægt er að fá er málskipunarvilla (e. syntax error). Python er frábrugðið öðrum forritunarmálum á þann hátt að málskipanin krefst þess að kóðinn sé settur upp á ákveðinn hátt. Líkja því má við að þurfa ekki að hafa greinamerki í huga þegar við ljúkum setningum heldur setjum við setningarnar okkar á réttan stað í samræðunum.

1.5.1 Uppsetning á kóða

Þessi kóðabútur er þannig uppsettur að allar línur byrja jafnlangt til vinstri, eins og hver setning í töluðu máli stendur hver lína fyrir sínu, ein og sér.

Kóðabútur 1.1: Réttur Python kóði

```
1 # Réttur Python kóði sem keyrist
2 4 + 8
3 5 + 6
4 breyta = 9 * 2
```

Þessi næsti kóðabútur hinsvegar er ekki nógu vel uppsettur, þar eru „setningar“ sem virðast hanga undir öðrum og vera þeim háðar.

Kóðabútur 1.2: Rangur Python kóði

```
1 # Illa skrifaður Python kóði sem keyrist ekki
2 4 + 8
3     5 + 6
4 breyta = 9 * 2
```

Svona inndrætti er einungis beitt ef lína á beinlínis að hanga undir línunni að ofan og tilheyrir henni. Þess vegna þarf að huga að því hvernig kóði er uppsettur. Í öðrum málum eru notuð greinamerki til að segja tölvunni að lína sé búin og að aðrar línur eigi að heyra undir eitthvað ákveðið samhengi en ekki í Python, þar er treyst á að forritarinn setji kóðann upp á máta sem hægt er að sjá að sé réttur. Dæmi um hvernig línur geta verið aðgreindar í öðrum málum:

Kóðabútur 1.3: Dæmi um annað mál sem er strangt tagað og með greinamerkjum

```
1 // Java
2 int i = 7;
3 i + 5;
4
5 // Þetta myndi líka ganga í Java en ekki í Python
6 int i = 7; i + 5;
```

Kóðabútur 1.4: Dæmi um annað mál sem byggir á afmörkuðu samhengi en með greinamerkjum

```
1 ; Lisp
2 (setq x 10)
3 (setq y 34.567)
4
5 (print x)
6 (print y)
```

Í þessum tveimur frábrugnu málum sem voru tekin sem dæmi var óþarfi að setja kóðann í mismunandi línur, því greinamerkin væru nóg til að aðgreina hverja línu fyrir sig. Hins vegar er það

góð venja að skrifa kóða sem er læsilegur öðru fólki. Í Java eru greinamerkin semikommur (;) en í Lisp eru línur og samhengi afmörkuð með svigum. Python byggist hinsvegar á því að forritarinn stilli öllu upp rétt með réttum inndrætti.

1.5.2 Gagnatýpur og lykilorð

Í Python eru nokkrar grunn gagnatýpur sem við munum kynna í þessari bók. Ástæðan fyrir því að þær eru kallað grunntýpur er sú að þær fylgja með Python uppsetningunni og notandinn getur beitt þeim í samræmi við það sem þær eru færar um, sem má skoða í skjölun Python <https://www.python.org/doc/>. Týpa eða tag er hugtak sem þýðir að hlutur sé af einhverri ákveðinni tegund sem má framkvæma ákveðnar aðgerðir á. Lesandi þekkir muninn á orðum og tölum úr daglegu tali og veit að hægt er að framkvæma mismunandi aðgerðir á þessum mismunandi týpum, eins og hægt er að skipta út hástöfum fyrir lágstafi í orðum en ekki tölum og hægt er að hefja tölur í veldi en ekki orð. Að sama skapi eru til aðgreinanlegar týpur sem tölvan kann skil á og leyfir ákveðnar aðgerðir á. Í fyrri hluta þessarar bókar verða gerð skil á tveimur talnatýpum (heiltölum og fleytitölum), strengjum, listum, orðabókum (einnig kallaðar hakkatöflur) og boolean gildum. Í seinni hlutanum bæstast svo við sett .

sett?

Lykilorð eru orð sem eru frátekin og birtast þau græn í Jupyter vinnubók. Hver gagnatýpa hefur eitt lykilorð og eru einnig nokkur innbyggð föll í Python, sem við kynnumst fljótlega, með frátekin orð. Forðast skal að yfirskrifa þessi lykilorð, en gerist það þá er auðvelt að laga það í Jupyter. Hver vinnubók hefur sinn kjarna til að vinna á og það eina sem þarf að gera í aðstæðum þar sem innbyggt orð er allt í einu farið að þýða eitthvað annað þá dugir að endurræsa kjarnann. Kjarninn í vinnubókinni er hvaða túlk eða þýðanda er verið að nota til þess að láta tölvuna skilja kóðann. Í okkar tilfelli erum við að nota Python 3.

2. Tölur og breytur

Í þessum kafla ætlum við að hefjast handa við að forrita. Það fyrsta sem við ætlum að gera er að kynna talnatýpum og keyra kóða eins og við værum að nota reiknivél. Við könnumst við reiknivélar og hvernig þær afgreiða röð aðgerða. Nú viljum við sannreyna að þær reiknaðgerðir sem við þekkjum séu til í Python og að þegar við keyrum kóðann okkar þá verði útkoman sú sama og við áttum von á. Við viljum líka geta geymt útkomuna okkar til að nota aftur seinna, til þess þurfum við breytur (e. variables).

2.1 Tölur - talnatýpur

Í Python eru í grunninn tvær týpur af tölum (en til eru tvær týpur af hvorri fyrir sig, sem snýr meira að minnisnotkun og er út fyrir svið þessarar bókar). Þær eru:

- **Heiltölur** - tölur sem eru ekki með neinum aukastaf. Á ensku eru þessar tölur kallaðar integers og er lykilorð þeirra því **int**.
- **Fleytitölur** - tölur sem eru með aukastaf, sem er fyrir aftan punkt (ekki kommu, fleytitölur eru oft kallaðar kommutölur á íslensku). Á ensku eru þessar tölur kallaðar floating point numbers og er því lykilorðið þeirra **float**.

Kóðabútur 2.1: Heiltölur og fleytitölur

```
1 # Heiltölur, enginn aukastafur
2 42
3 100000
4 -139
5
6 # Fleytitölur, aukastafur/ir fyrir aftan punkt
7 4.0
8 3.1415926
9 -100.98
```

2.2 Reikniaðgerðir og tákni

Grunnreikniaðgerðir eru nokkrar sem við könnumst við úr grunnskóla en aðrar eru framandi og við skulum skoða aðeins betur.

Táknin eru flest eins og á reiknivélum $+$, $-$, $*$, $/$ en þar að auki er annars konar deiling sem er táknuð með tveimur deilimerkjum $//$, veldishafning er táknuð með tveimur margföldunarmerkjum $**$, og svo er leifareikningur táknaður með $\%$. Heiltöludeiling og leifarreikningur eru líklega ný á nálinni fyrir flestum lesendum og því allt í lagi að útskýra þær aðgerðir aðeins nánar. Þessar aðgerðir eru einmitt mjög skyldar í raun. Deilingin segir okkur hversu oft ein tala gengur upp í aðra þar sem útkoman er heil tala (eða fleytitala með 0 sem eina aukastafinn), okkur er sama um afganginn sem verður eftir. Í þessari deilingu er svarið 2 við bæði $5//2$ og $4//2$. En í leifarreikningnum viljum við eingöngu vita hver er afgangurinn þegar heiltöludeilingu er beitt svo $5\%2$ væri 1, því það er einn í afgang þegar fimm er deilt með tveimur. Og það er 0 í afgang þegar fjórum er deilt með tveimur svo $4\%2$ er 0.

Í eftirfarandi dæmum í kóðabút 2.2 er vert að draga fram nokkur atriði sem eru ekki augljós byrjanda. Það fyrsta er að myllumerkið (#) þýðir að allt sem kemur fyrir aftan það er *athugasemd*, athugasemdir eru engöngu til að gera kóða læsilegri fyrir fólk, þær eru hunsaðar af tölvunni þegar hún breytir kóðanum í eitthvað sem hún skilur. Eins og sést í línu merktri númer 20 er athugasemdin svo löng að hún birtist okkur sem tvær línur en hún er í keyrslu tölvunnar álitin ein heild línu 20. Þess vegna þurfum við ekki að hafa áhyggjur af þessum inndrætti sem birtist, hann er í rauninni ekki til staðar þar sem þessi hluti textans er ein heild. Einnig eru þarna bil á milli talna fremst í línu og tákna, það er líka til að gera kóða læsilegri, bilin mega bara ekki vera fremst í línunni enn sem komið er. Athugasemdir í kóða eru mjög mikilvægur hluti af skjölun kóða og ættu öll sem vilja tileinka sér forritun að venja sig á að skrifa athugasemdir. Í fyrstu erum við ekki að skrifa flókinn kóða svo athugasemdirnar segja okkur ekki mikið, en þegar kóðinn er ekki augljós eða lausn á verkefni ekki augljós er gott að skrifa athugasemdir. Flest allir kóðabútar eru skjalaðir með athugasemdum til að gera þá læsilegri því allur kóði í bókinni er skrifaður fyrir fólk til að skilja. Kóði sem þið komið til með að skrifa seinna meir á einnig að vera ykkur sjálfum skiljanlegur þegar þið komið að honum seinna. Því er gott að venja sig strax á að skrifa lýsandi athugasemdir.

Kóðabútur 2.2: Reikniaðgerðir

```

1 # Samlagning framkvæmd með +
2 # Þegar eftirfarandi kóði er keyrður ætti útkonan að vera 10
3 6 + 4
4
5 # Frádráttur framkvæmdur með -
6 # Þegar eftirfarandi kóði er keyrður ætti útkonan að vera 10
7 14 - 4
8
9 # Margföldun framkvæmd með *
10 # Þegar eftirfarandi kóði er keyrður ætti útkonan að vera 10
11 10 * 2
12
13 # Deiling framkvæmd með /
14 # Athugið að þetta er fleytitöludeiling sem skilar nákvæmu svari
15 # Þegar eftirfarandi kóði er keyrður ætti útkonan að vera 10.0
16 60 / 6
17
18 # Heiltöludeiling framkvæmd með //
19 # Athugið að þessi deiling er frábrugðin þeirri sem þið kannist við
20 # Hér viljum við vita hversu oft, heil tala, ein tala gengur upp í aðra og okkur er sama
   um afganginn
21 # Þegar eftirfarandi kóði er keyrður ætti svarið að vera 10
22 177 // 17
```



```

23
24 # Veldishafning framkvæmd með **
25 # Hér er mikilvægt, eins og með deilinguna, að hafa í huga hvor talan kemur á undan.
26 # Fyrst kemur talan sem hefur á í veldi og svo kemur talan sem er veldisvísirinn
27 # Þegar eftirfarandi kóði er keyrður ætti svarið að vera 9
28 3 ** 2
29
30 # Leifareikningur framkvæmdur með % (e. modulus)
31 # Þetta er eitthvað alveg nýtt og framandi, en þó ekki óskiljanlegt
32 # Það sem þetta reiknar er hversu mikil leif eða afgangur er eftir þegar heiltöludeilingu
   er beitt.
33 # Þegar eftirfarandi kóði er keyrður ætti svarið að vera 7
34 177 % 17

```

Í öllum þessum dæmum var verið að vinna með heiltölur, þó var útkoman úr deilingunni (stundum kölluð fullkomin deiling) fleytitala. Hvað gerist ef þessir sömu útreikningar eru gerðir með fleytitölum? Ef við myndum skipta út hverri tölu fyrir sig og setja í staðinn sömu tölu með .0 fyrir aftan þá yrðu útkomurnar þær sömu nema fleytitölur. En hvað gerist ef við breytum aðeins fyrri tölunni en ekki seinni tölunni? Þá ertu að nota ólíkar týpur og slíkt er vandmeðfarið, en í þessu tilviki er það í lagi þar sem Python gerir þá ráð fyrir að það sé í lagi að reikna allt með fleytitölum og framkvæmir reikninginn eins og þú hafir verið að beita fleytitölum í hvívetna og niðurstaðan verður þá að sjálfsögðu fleytitala.

2.3 Breytur

Nú höfum við séð hvernig má keyra kóða einfaldlega eins og í reiknivél. Höldum okkur við samlikinguna um reiknivélina til að útskýra breytur. Á hefbundinni reiknivél sem notuð er í stærðfræðitíma í framhaldsskóla er takki sem á stendur ANS. Það stendur fyrir answer og ef ýtt er á hann getur vélin geymt síðasta gildið sem hún gaf sem svar og unnið svo með það til að gefa næsta svar. Flottari vélar geta svo geymt nokkuð mörg svör en það er útfyrir gagnsemi þessarar samlikingu. Þegar ýtt er á þennan takka er minnisvæði í reiknivélinni tekið frá og skrifað er í það gildi, sem er svo sótt þegar ANS er notað í útreikningi. Að sama skapi má láta Python úthluta minnisvæði í tölvunni fyrir þær breytur sem þið viljið geyma. Munurinn er sá að þið nefnið sjálf hvað minnisvæðið er merkt sem, eruð ekki bundin við að nota ANS og að þið eruð svo gott sem með óteljandi minnisvæði.

Að gefa minnisvæði merkingu og gildi er gert með *gildisveitingu*. Gildisveiting þýðir að nú er einhver ákveðinn merkimiði kominn með eitthvað til að geyma. Sjáum einfalt dæmi um þetta.

Kóðabútur 2.3: Breytur kynntar

```

1 # Hér er ég að fara að búa til breytu sem heitir val
2 val = 5
3
4 # Þegar ég keyri línuna fyrir ofan segi ég vélinni að hafa aðgengilegt minnisvæði sem ég
   get notað með því að skrifa orðið val, og settu í það svæði gildið 5.
5
6 # Svo ég er að veita breytunni val gildið 5, þess vegna er það kallað gildisveiting.
7
8 # Svo get ég notað breytuna mína
9 # þegar þetta er keyrt fæst svarið 10
10 val + 5

```

Ef þú prófar þig áfram við að búa til breytur gætir þú rekist á svolítið sem hefur ekki gerst áður í vinnubók, að þegar sella inniheldur eingöngu gildisveitingu og er keyrð þá „gerist ekkert“. Þetta finnst mörgum mjög skrítið því þau vilja fá einhverja útkomu. En útkoman er sú að þú sagðir

vélinni að geyma þetta, þú sagðir henni ekki að gera neitt annað.

Breytur eru skilgreindar vinstra megin við jafnaðarmerki í Python. Eins og það væri lesið, val fær gildið 5. Það væri lítið vit í því að hafa það öfugt, 5 er núna jafngilt val. Það sem við værum þá að segja tölvunni að í hver sinn sem hún vill nota heiltöluna fimm þá á hún að hætta við að nota töluna sjálfa og í staðinn vísa eingöngu í það sem er í minnissvæði merktu val. Það er alls ekki það sem við viljum.

Nokkrar reglur í nafnavali á breytum, þetta vill vefjast fyrir sumum en lærist fljótlega:

1. Kóðalítingin á breytuheitinu má ekki vera annað en venjulegi liturinn fyrir kóða, þannig að ef nafnið fær áherslumerkingu (annan lit) er það ekki löglegt breytuheiti. Áherslulítingin í númeruðu kóðabútnum í þessari bók er marklaus því hún er mjög frumstæð. Dæmi um það sem fær áherslulítingu eru frátekinn lykilorð og tölustafir.
2. Breytuheitið ætti ekki að innihalda séríslenskan staf (það er löglegt í jupyter vinnubókum en er hrikalega slæmur ávani því það er ekki löglegt allsstaðar).
3. Breytuheitið má ekki innihalda bil.

Nokkur tilmæli um breytunöfn með tilliti til nafnavenja í Python:

1. Breytuheiti byrja á litlum staf.
2. Ef það þarf að gera löng breytuheiti er venjan að nota snákaframsetningu (e. snake casing) sem felur í sér að gera niðurstrik á milli orða, dæmi `thetta_er_langt_nafn_a_breytu`. Annars er til kamelframsetning (e. camel casing) sem felur í sér að annað hvert orð er með stórum staf, dæmi `thettaErLikaLangtBreytuheiti`. Hvort sem þið endið á að nota meira, haldið ykkur bara við annað þeirra.
3. Breytuheiti eiga að vera lýsandi. Ef ég væri að reikna hliðar í þríhyrningi væri gott að eiga breyturnar `a`, `b` og `c`. En ef ég væri að búa til reiknirit sem býr til tölvuleikjapersónu af handahófi með því að velja tilviljanakennt nafn, aldur og starf þá væru breytuheitin `a`, `b` og `c` alveg glötuð því þegar ég kæmi aftur að kóðanum mínum myndi ég ekki hafa hugmynd um hvað `a`, `b` og `c` væru. Betra væri að breyturnar hétu nafn, aldur og starf.

Kóðabútur 2.4: Dæmi um gildisvætingar réttar og rangar

```

1 # Hér er ég að fara að búa til breytu sem heitir val
2 val = 5
3
4 # Hér er ég ekki að búa til breytu sem heitir val heldur er ég að segja að talan fimm er
   ekki lengur til sem heiltala heldur gæti hún vísað í hvað sem er sem er geymt í
   minnissvæði merktu val, ólöglegt.
5 5 = val
6
7 # Hér bý ég til breytu sem heitir heiltala sem fær gildið 0
8 heiltala = 0
9
10 # Hér yfirskrifa ég lykilorðið fyrir týpuna heiltala og læt það innihalda 0
11 int = 0
12 # Þetta er harðbannað og ef þetta gerist er ekki nóg að þurrka þetta út og keyra aftur,
   nú þarf að endurræsa kjarna vinnubókarinnar.
13
14 Gott nafn = 1.0
15 # Þetta er ekki bara bannað vegna bilsins á milli orðanna „Gott“ og „nafn“ heldur er þ
   að líka ljótt því að það byrjar á stórum staf
16
17 3_litlar_mys = 3
18 # má ekki byrja á tölustaf eða tákni
19
```

```

20 utreiknud_laun_eftir_skatt = 0.65 * laun
21 # frábært, lýsandi og gott breytuheiti (hér er þó gert ráð fyrir að vélin þekki breytuna
    laun)

```

Nú þegar við höfum séð hvernig má skilgreina breytu viljum við vita hvernig á að nota þessa breytu. Ef við snúum okkur aftur að reiknivélasamlíkingunni um ANS takkann þá ætti kóðabútur 2.5 að geta sýnt með eðlislægum hætti hvernig breytur nýtast. Fyrst segi ég vélinni hvað það er sem ANS vísar á, svo segi ég vélinni að mig langar til þess að búa til nýja breytu sem á að byggja á því sem ANS inniheldur. Í þessum kóðabút er svo haldið áfram með þessa afleiddu breytu og önnur afleidd breyta búin til útfrá henni. Það sem gerist svo í endann er sambærilegt við það að ýta á „=” takkann á reiknivélinni. Takið eftir að þarna er notuð ný framsetning sem við höfum ekki séð áður, þarna stendur `print` með svigum fyrir aftan og inni í svigunum er breytan okkar. Ef þessi kóðabútur er keyrður þá kemur á *staðalúttak*¹ það gildi sem breytan `x` inniheldur. Ef þar hefdi staðið `print(halft_x)` hefðum við fengið svarið sem er geymt í breytunni `print(halft_x)`.

Kóðabútur 2.5: Að nota breytu

```

1 # Hér framkvæmi ég einhvern útreikning sem ég geymi í breytunni ANS
2 ANS = 5**2 + (4+8.9)**2
3
4 # Segjum að þetta hafi verið endapunkturinn í löngu algebrudæmi og nú veit ég hvað y er,
    og get þá nýtt það til að finna x eins og verða vill svo oft í stærðfræði að x sé
    týnt. Gefum okkur að x = 3 * y og því fæst
5 x = 3 * ANS
6
7 # Nú ef við viljum reikna eitthvað út með x eigum við það til í minnissvæði merktu x með
    réttu gildi. Til dæmis með því að búa til breytu fyrir hálft x.
8 halft_x = x/2
9
10
11 # Nú langar okkur til að vera viss um að við séum við vitrænt svar svo við biðjum tölvuna
    um að segja okkur hvað er geymt í breytunni x.
12 print(x)
13 a = "texti í streng"
14 'strengur'
15 "annar strengur"

```

Við megum beita `print` skipuninni óspart og hvetur höfundur til þess að lesandi venji sig á að skoða úttakið sitt í hverju þrepi áður en leitað er hjálpar til annarra. `Print` er *fall*, við skoðum föll nánar í kafla 8 en þangað til munum við kynnast nokkrum innbyggðum föllum eins og `print()`.

Núna höfum við séð tvær týpur, heiltölur og fleytitölur. Breyta getur innihaldið hvernig týpu sem er. Þá þurfum við að athuga hvað má gera við breyturnar okkar. Hingað til höfum við eingöngu skoðað reikniðgerðir sem eru framkvæmdar með kunnuglegum táknum, við höfum ekki verið að beita neinum innbyggðum *aðferðum* á tölurnar okkar. Við sjáum það gert í kafla 3 þegar við skoðum hvernig megi vinna með texta.

Að því sögðu þá þurfum við að skoða breytur nokkuð betur áður en við förum að beita þeim á skilvirkan hátt. Við erum búin að skoða reiknivirkja og gildisveitingu, og nú ætlum við að skoða *reiknivirkjagildisveitingu* þar sem við uppfærum gildi í breytu með því að nota reiknivirkja eins og `+` eða `-` með gildisveitingu `=`. Þetta sést betur í kóðabút

Kóðabútur 2.6: "Reiknivirkjagildisveiting"

```

1 # ég ætla að telja nemendur inn í stofuna
2 # ég byrja með 0 nemendur

```

¹ Þann stað sem texti myndi prentast þegar forritið er notað, hvort sem það er á skjá eða beint á pappír úr prentara eða eitthvað allt annað. Kannski verður úttaki varpað beint inn í heilann á forriturum einhvern tíma?

```
3 nem = 0
4
5 # svo sé ég fyrsta nemandann minn
6 nem = nem + 1
7
8 # þá uppfæri ég gildið sem nem breytan inniheldur og er hún núna það sem hún var (0) + 1,
  ef ég keyrði þessa línu aftur væri það orðið að (1) + 1 og svo koll af kolli eftir þ
  ví sem ég keyri þessa línu oftari
9
10 # önnur leið til að skrifa þetta er með reiknivirkjagildisveitingu
11 nem += 1
12
13 # þá uppfæri ég gildið nem um það sem hún var + 1
14
15 #segjum að nemendur koma inn í stofuna í þörum þá væri formúlan svona:
16 nem += 2
17
18 # þá uppfærir gildið í nem um +2 í hvert sinn sem þessi kóðalína er keyrð
19
20 # Ef ég væri að reikna stofnstærð á bakteríum sem tvöfaldast á hverjum klukkutíma gæti ég
  gert það svona:
21 stofn_staerd = 30
22 stofn_staerd *= 2 # og keyrt svo þessa línu fyrir hvern klukkutíma
23
24 # Tökum annað dæmi, byrjum á að skoða hvernig megji geyma útreikninga í mismunandi breytum
25 thusund = 1000
26 fimm_hundrud = thusund/2
27 tvo_hundrud_og_fimmtiu = fimm_hundrud/2
28
29 # ef ef mér er sama um breytuna thusund og vil þess í stað bara halda utan um alla
  upphæðina mína í einni breytu og helminga hana tvisvar þá get ég gert þetta
30 allt = 1000
31 allt /= 2
32 allt /= 2
33
34 #nú er allt orðið að 250
35
36 # ég má líka nota aðrar breytur í uppfærslunni minni
37 # hér ætla ég að reikna út hver hækkan launa yrði milli ára ef ég fengi alltaf 2%
  launahækkun
38
39 laun = 100
40 verdbolga = 0.02
41 laun *= 1 + verdbolga # þetta má svo keyra endurtekið til að skoða fram í tímann
```

Hér sést að það er gagnlegt og fljótlesið þegar það á að uppfæra gildi á breytu að gera það með því að nota reiknivirkjann með gildisveitingunni. Í kóðabútnum að ofan sést að athugasemdir eru skrifaðar inni í línunum líka, það er stundum gagnlegt að skrifa stuttar athugasemdir inni í kóða með þessum hætti en betra er þó að skjala hann skilmerkilega efst við viðeigandi kóðabút. Allar athugasemdir eru hunsaðar af vélinni og því hefur allt sem er fyrir aftan # merkið, hver sem það er, engin áhrif á útkomuna.

3. Strengir

Til þess að geta sýnt og notað texta þarf gagnatýpu til að halda utan um hann. Í flestum forritunarmálum, og Python er ekki undantekning, eru gögn af þeirri týpu kölluð **strengir**. Lykilorð fyrir þessa týpu er **str**.

3.1 Strengir skilgreindir

Til þess að afmarka texta og segja vélinni að fara með hann sem af týpunni strengur þarf að nota tákn. Við þurftum ekki að gera það þegar við skrifuðum tölurnar en nú, og seinna, munum við þurfa sér tákn til þess að segja vélinni gögn af hvaða týpu hún er að vinna með.

Táknin sem skilgreina strengi eru gæsalappir. Einfaldar eða tvöfaldar.

Kóðabútur 3.1: "Strengir skilgreindir

```
1 # Fyrsti strengurinn okkar
2 "halló"
3
4 #strengur geymdur í breytu
5 textinn_minnt = "halló ég má skrifa mörg orð inn í þessar gæsalappir"
6
7 einfaldar_gæsalappir = 'ég má líka skrifa innan einfaldra gæsalappa'
8
9 thetta_virkar_ekki = 'gæsalappirnir þurfa að passa saman'
10
11 # og ef ég vil skrifa mjög langan texta nota ég þrjár gæsalappir
12 langi_textinn_minnt = ''' ég má skrifa eins langa setningu hér og ég vil því að þetta
    verður alltaf álitid sem ein lína, hins vegar ef ég nota öðruvísi gæsalappir og
    langar að gera kóðan læsilegan er hægt að brjóta hann upp án þess að nota þessar þ
    reföldugæsalappir, við sjáum það eftir smá'''
```

Í sumum forritunarmálum er munur á því að nota einfaldar og gæsalappir, þar sem einfaldar eru notaðar fyrir staka stafi (sér gagnatýpa) og tvöfaldar fyrir strengi. En það er enginn raun munur á því hvernig Python meðhöndlar þær.

3.2 Strengir og reikniaðgerðir

innubók inni í
kóðabút

Við erum búin að sjá að það megi leggja tölur saman og margfalda þær. Nú ætlum við að skoða hvaða reikniaðgerðir er hægt að framkvæma með strengi og hvaða áhrif það hefur.

Kóðabútur 3.2: "Strengir og reikniaðgerðir"

```
1
2 # Reikniaðgerðirnar sem við þekkjum eru +, -, *, /, //, **, og %
3 # Lesandinn er hvattur til þess að gera prófanir á þessu í vinnubók upp á eigin spýtur
4 # Með því að skilgreina streng og reyna að nota reikniaðgerð á hann með tölum eða öðrum
   strengjum
5
6
7 # Gerum ráð fyrir að þessar prófanir hafi átt sér stað og niðurstaðan sé sú að þær
   aðgerðir sem hægt er að framkvæma eru + og *
8 # En hvað gerist þegar við notum þær?
```

Þegar við notum + til að setja saman strengi þá erum við að beita *samskeytingu* (e. concatenation). Samskeyting þýðir að einum streng er bætt við fyrir aftan annan streng. Það skiptir máli hvor er fyrir framan: "halló" + "bless" verður að "hallóblessen" "bless" + "halló" verður að "blesshalló".

Kóðabútur 3.3: "Samskeyting strengja"

```
1
2 strengur_a = "þetta er a strengurinn minn"
3 strengur_b = " og þetta er b strengurinn minn"
4
5 # Nú get ég sameinað þessa strengi með því að setja annan þeirra fyrir aftan hinn
   sameinadir_a_og_b = a + b
6
7
8 # ef við prentum út strenginn fáum við
9 "þetta er a strengurinn minn og þetta er b strengurinn minn"
10 #takið eftir að það er bil á milli strengjanna, það er eingungis vegna þess að b
   strengurinn er skilgreindur þannig að fyrst kemur bil fremst í strengnum
11
12 # röðin skiptir máli þegar strengir eru sameinaðir svona
   sameinadir_b_og_a = b + a
13
14
15 # þetta útprentað skilar okkur:
16 " og þetta er b strengurinn minnþetta er a strengurinn minn"
17
18 # Takið eftir að þarna er ekkert bil á milli strengjanna.
19
20 # Á þessu er hægt að svindla:
21 fyrsta_nafn = "Valborg"
22 seinna_nafn = "Sturludóttir"
23 fullt_nafn = fyrsta_nafn + " " + seinna_nafn
24
25 # Þarna sameinaði ég þrjá strengi þar sem ég vissi að hvorugur strengjanna minna
   innihéldi bil ákvað ég að setja það á milli með auka samskeytingu.
```

Þegar við notum * til þess að margfalda streng erum við að *lengja* (e. multiply) hann. Strengjalenging virkar þannig að þú tilgreinir hversu oft, í heilum tölum, þú vilt að strengurinn sé endurtekinn.

Kóðabútur 3.4: "Strengjalenging"

```
1 eitt_ord = "kex"
2 eitt_ord*3
3
```

```
4 #skilar okkur  
5 "kexkexkex"
```



4. Listar

4.1 Gagnagrindur



5. Segðir og skilyrðissetningar

5.1 Segðir




6. Lykkjur

6.1 while - á meðan eitthvað er satt



7. Orðabækur

7.1 Lyklar og gildi



8. Föll

8.1 Tilgangur falla

8.2 Að skrifa föll

8.3 Viðföng

8.4 Skilagildi

8.5 Lokun

8.5.1 Descriptions and Definitions

Name Description

Word Definition

Comment Elaboration

9. In-text Elements

9.1 Theorems

This is an example of theorems.

9.1.1 Several equations

This is a theorem consisting of several equations.

Theorem 9.1.1 — Name of the theorem. In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$||\mathbf{x}|| - ||\mathbf{y}|| \leq ||\mathbf{x} - \mathbf{y}|| \quad (9.1)$$

$$||\sum_{i=1}^n \mathbf{x}_i|| \leq \sum_{i=1}^n ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \quad (9.2)$$

9.1.2 Single Line

This is a theorem consisting of just one line.

Theorem 9.1.2 A set $\mathcal{D}(G)$ is dense in $L^2(G)$, $|\cdot|_0$.

9.2 Definitions

This is an example of a definition. A definition could be mathematical or it could define a concept.

Definition 9.2.1 — Definition name. Given a vector space E , a norm on E is an application, denoted $||\cdot||$, E in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \Rightarrow \mathbf{x} = \mathbf{0} \quad (9.3)$$

$$||\lambda \mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \quad (9.4)$$

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \quad (9.5)$$

9.3 Notations

Notation 9.1. Given an open subset G of \mathbb{R}^n , the set of functions φ are:

1. Bounded support G ;
2. Infinitely differentiable;

a vector space is denoted by $\mathcal{D}(G)$.

9.4 Remarks

This is an example of a remark.



The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

9.5 Corollaries

This is an example of a corollary.

Corollary 9.5.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

9.6 Propositions

This is an example of propositions.

9.6.1 Several equations

Proposition 9.6.1 — Proposition name. It has the properties:

$$||\mathbf{x}|| - ||\mathbf{y}|| \leq ||\mathbf{x} - \mathbf{y}|| \quad (9.6)$$

$$||\sum_{i=1}^n \mathbf{x}_i|| \leq \sum_{i=1}^n ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \quad (9.7)$$

9.6.2 Single Line

Proposition 9.6.2 Let $f, g \in L^2(G)$; if $\forall \varphi \in \mathcal{D}(G)$, $(f, \varphi)_0 = (g, \varphi)_0$ then $f = g$.

9.7 Examples

This is an example of examples.

9.7.1 Equation and Text

■ **Example 9.1** Let $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ and denoted by: $x^0 = (1, 1)$; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \quad (9.8)$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \varepsilon\}$ for all $\varepsilon \in]0; 5/2 - \sqrt{2}[$. ■

9.7.2 Paragraph of Text

■ **Example 9.2 — Example name.** Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. ■

9.8 Exercises

This is an example of an exercise.

Exercise 9.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

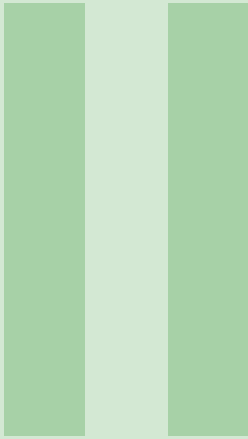
9.9 Problems

Problem 9.1 What is the average airspeed velocity of an unladen swallow?

9.10 Vocabulary

Define a word to improve a students' vocabulary.

Vocabulary 9.1 — Word. Definition of word.



Part Two

10	Presenting Information	41
10.1	Table	
10.2	Figure	
	Bibliography	43
	Articles	
	Books	
	Index	45

10. Presenting Information

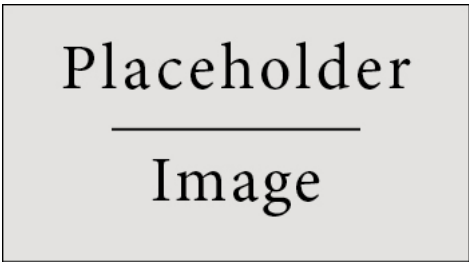
10.1 Table

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Tafla 10.1: Table caption

Referencing Table 10.1 in-text automatically.

10.2 Figure



Mynd 10.1: Figure caption

Referencing Figure 10.1 in-text automatically.



Bibliography

Articles

[1] James Smith. “Article title”. Í: 14.6 (mar. 2013), blaðsíður 1–8.

Books

[2] John Smith. *Book title*. 1. útgáfa. Bindi 3. 2. City: Publisher, jan. 2012, blaðsíður 123–200.

Atriðisorðaskrá

C

Citation	8
Corollaries	10

D

Definitions	9
-------------------	---

E

Examples	10
Equation and Text.....	10
Paragraph of Text	11
Exercises	11

F

Figure	15
--------------	----

L

Lists	8
Bullet Points	8
Descriptions and Definitions	8
Numbered List.....	8

N

Notations	10
-----------------	----

P

Paragraphs of Text	7
Problems	11
Propositions	10
Several Equations	10
Single Line	10

R

Remarks	10
---------------	----

T

Table	15
Theorems	9
Several Equations	9
Single Line	9

V

Vocabulary	11
------------------	----