Kaare Erlend Jørgensen
Stein Alexander Dahl

PYTHON PROGRAMMING: A VISUAL JOURNEY FOR THE BEGINNER

WITH SIMPLE APPLICATIONS IN MATHEMATICS

Second Edition

Jorgensen Matematiske Ressurser

© Second Edition, Jørgensen Matematiske Ressurser 2021

ISBN 978-82-692219-5-4

All rights reserved. No part of this book may be used, transmitted or reproduced in any manner whatsoever without written permission except in the case of brief quotations embodied in critical articles and reviews. For information, address the copyright managing authority (www.kopinor.no).

This book has its own webpage: www.kaareskokebok.no

Any enquiry about the present work may be adressed using: kaare@kaareskokebok.no

Illustrations and graphical production: Kaare E. Jørgensen & Stein A. Dahl
Fonts: Computer Modern 10/12, pdflatex/TikZ/tcolorbox

Acknowledgements

> "Of all the characteristics needed for both a happy and morally decent life, none surpasses gratitude. Grateful people are happier, and grateful people are more morally decent."
>
> *Dennis Prager*[1]

These wise words of Dennis Prager reminds me that I have a lot to be grateful for, and I owe gratitude to many people. I want to begin with thanking my dear wife, Inna Jørgensen, for never loosing faith in my book-project. She was always willing to step up to manage our home and our children. My co-author, Stein A. Dahl, has been indispensable in solving technical and design problems. He has authored several parts of the book, and gone through the academics with a precision few can match. I want to give thanks to my mother, Turid Jørgensen, for being a great support during busy days, helping out with the kids and being a great listener. I also want to thank my father, Åge J. Marthinsen, for all the important values he taught me such as hard work, discipline and perseverance.
Further I want to thank Ellen Egeland Fløe for recommende me as a programming-instructor to other teachers. Through BRO AOF I was able to travel around Norway, giving programming seminars to teachers, which all gave important knowledge and experience on how students ideally should learn programming. A thank you goes to Nina Sundmark for useful feedback early on. I give thanks to Thomas F. Sturm[2], who has been of great help with the design of all the colored boxes throughout the book. His impressive manual of 525 pages has been a blessing of discovery and learning. A thanks goes to Tantau[3] for his help with graphs and figures produced with Tikz. His manual of 1318 pages is truly a piece of art. I want to thank Master Books Publisher for giving me permission to use examples and exercises from the great geometry book by Harold R. Jacobs[4]. Finally, a thanks to Inga Stener Olsen for useful feedback to the introduction, to Audun Åby for the suggestion of creating a thourough index, and to Isabella Dahl for the final round of proofreading.

Kaare Erlend Jørgensen

---

[1] Prager, Happiness is a serious problem [6]
[2] Sturm, The tcolorbox package [12]
[3] Tantau, The Tikz and PGF manual [13]
[4] Jacobs, Geometry: Seeing, Doing, Understanding [4]

Contents

Introduction

What is this book about?

This book teaches you the basic concepts of programming in the language *Python*. The first part of the book takes you on a visual tour through the basic programming concepts. The second part of the book applies these concept to explore selected mathematical topics. We teach you the required math as we go along, so math knowledge is not a prerequisite to study programming with this book.

Who is this book for?

This book is for you who have never programmed before. You don't even know what a program or programming is? That's perfect. This book is for you. The book is also for students who need to use programming in math class. Recommended age-group is 12+. Teachers may find this book very useful, because the knowledge of good ways to teach programming from scratch is scarce in schools around the world. This book is written also for the classroom, specifically the math classroom. We have also written this book with parents in mind. If you are a parent who wants to help your child with programming, but you never learned programming yourself, then this book is for you too! For teachers, we recommend you read our appendix C to understand the learning principles behind this book.

How should I read this book?

This book is designed to teach you programming in a linear fashion. Hence, it is very important that you read the bok in order. Chapter 1, Chapter 2, Chapter 3, and so on. The skills you learn in Chapter 1 are applied in Chapter 2. The skills you learn in the first four chapters, are applied in Chapter 5, and so on. After you completed the first five chapters, you will have a good grasp of the concepts used in chapters 6 through 10. Therefore it is possible to jump between chapters after you read through the first part of the book.

Where Can I Write the Code?

You do not need to install any software to get going with programming. All the examples in this book are written on the website `replit.com`. Here we use two versions of Python: *Python (with Turtle)* and *Python3*.

The following two alternative websites are worth mentioning:

- *Python Turtle*: `trinket.io/features/pygame` and `pythonandturtle.com/turtle`. Note that on these two websites, all Turtle-programs must finish with the command `done()`.
- *Python3*: `trinket.io` and `pythonandturtle.com`. On the first mentioned website, choose New Trinket » Python3 to get started.

What is new in the second edition?

The second edition contains many structural and design improvements, in addition to the following:

- All chapters now end with both a **summary** and **exercises**.
- Most chapters are extended with many **new and exciting exercises**.
- There are now **complete solutions** to almost all exercises. The solutions are in the back of the book.
- We have included a **list of figures**, a rich **index** over all Python-words and **keyword list** over all a lot of the book's English words.

What do the symbols in the margin mean?

133 In the margin you will often see a page number in a purple box. This tells you on which page you will find the solution to the current exercise/section.

The star-symbol means that this exercise is considered challenging, and that you will not find a solution in the back of the book. We want our students to feel a sense of great accomplishment, when they master these challenges.

The turtle-symbol means that the code must be written with the language *Python Turtle*. Go to `replit.com`, create a new file and select the language *Python (with Turtle)*. If this symbol is not present, the code is to be written in the language *Python*. You can also use these links: `replit.com/new/python_turtle` og `replit.com/new/python3`.

Good luck on your programming-journey!

`Kaare Erlend Jørgensen`

`Stein Alexander Dahl`

x

PART I

The Basics

CHAPTER 1

The First Instructions

1.1 The Creation

Figure 1.1: *A real turtle.*

We will start by telling our computer to create a turtle. We call our turtle Tom, and he's going to do exactly what we instruct him to do. Tom does not understand English, only Python: this book's programming language.

a) Open the webpage `replit.com`. Create a new user, and log in. We recommend the browsers *Brave* and *Chrome*.

b) Create a new file by clicking + on the top right. Choose the language "Python (with Turtle)", and then select "Create repl". See figure 1.2.

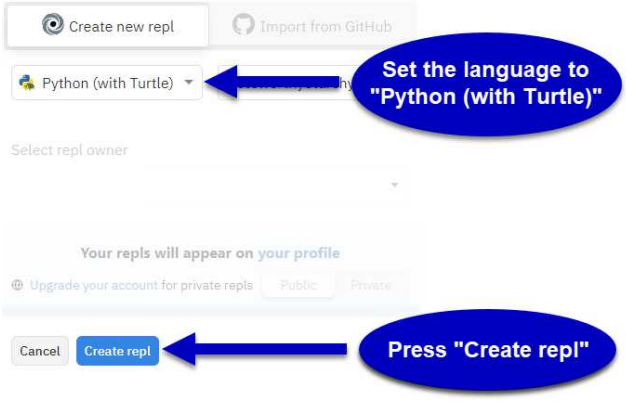


Figure 1.2: *Create a new file on* `replit.com`.

c) Write the following code to create Tom the Turtle.

Code

```
1 from turtle import *
2 shape("turtle")
```

Figure 1.3: *The result of the code.*

To run the program, press Run ▶ , or press the keys Ctrl + Enter (Mac: Cmd + Enter) on the keyboard. Do you see a turtle now, like figure 1.3 shows, or did you get an error message?

d) If you do see the turtle, move to e) now. If you got an error message, check below and see if you recognize your mistake. Correct your code and run the program again.

Correct

```
from turtle import *
```

Wrong

```
from Turtle import *
from turtle import*
from turlte import *
from import turtle *
```

Correct

```
shape("turtle")
shape('turtle')
```

Wrong

```
shape 'turtle')
shape(turtle')
shape(`turtle`)
shape('Turtle')
```

Notice that you can write `shape("turtle")` or `shape('turtle')`.

e) Congratulations! You have now created Tom the Turtle and you're ready to give him instructions.

1.2 The First Small Steps

117

Code

```
from turtle import *
shape('turtle')
speed(1)
forward(50)
left(30)
```

Tom the Turtle understands the instructions `forward` and `left`. He does not understand "walk over there!" or "turn hard left now!".

a) Read the code above. What do you think the result of the code will be?

b) Copy the code, and run the program. Press Run ▶ or Ctrl+Enter (Mac: Cmd+Enter) to run the program.

c) Let `forward(50)` and `left(30)` switch lines. Guess the result, and then run the program.

d) Add the following instructions at the bottom of the code. Guess the result, and then run the program.

```
left(-30)
forward(50)
```

e) Let `left(30)` and `left(-30)` switch lines. Guess the result, and then run the program.

f) Finally, finish writing the code so that the result is figure 1.4.

Figure 1.4: *Drawing with lines.*

In the back of the book you will find solutions to the exercises. The solution to this section, section 1.2, you will find on page 117. This pagenumber is marked in the margin as 117.

1.3 Triangle Recipe

117

Code

```
1 from turtle import *
2 shape('turtle')
3 color('blue')
4 forward(75)
5 left(60)
6 forward(75)
```

Recipe

```
Create the turtle
Set the shape to a turtle
Set the color to blue
Walk 75 steps forward
Turn 60 degrees left
Walk 75 steps forward
```

Writing code can be compared to writing a recipe. The code is in Python, whereas a recipe is in English.

a) Read the code above. Guess the result.

b) Write the code, and run the program. Press Run ▶ or Ctrl+Enter (Mac: Cmd+Enter) to run the program.

c) Color Tom with your favourite color[1] by changing `color('blue')` to another color. For example `color('brown')` .

d) Change `left(60)` to `left(120)` , and run the program.

e) Add the lines `left(120)` and `forward(75)` in the correct order so that the program produces figure 1.5.

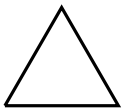


Figure 1.5: *An equilateral triangle.*

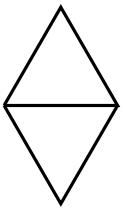


Figure 1.6: *Two equilateral triangles.*

f) Add the line `hideturtle()` to the bottom of the code.

g) Expand the program so that the result becomes figure 1.6.

h) Compare your solution to the one you find in the book on page 117. This pagenumber is also marked in the margin as 117 .

[1] On the web page trinket.io/docs/colors, you can select a color and see the name of that color in Python Turtle.

1.4 Square and Rectangle

Code

```
1 from turtle import *
2 shape('turtle')
3 forward(200)
4 left(90)
5 # And so on ...
```

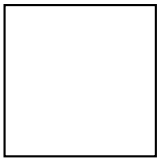


Figure 1.7: *A square.*

On line 5 we have made a *comment* by writing # before the text. Comments are useful to explain the meaning of the code.

a) Write the code above, and run the program.

b) Continue writing the instructions [2] `forward(200)` and `left(90)` enough number of times and in the right order, until you finish drawing a square. See figure 1.7.

c) Add the line `speed(7)` directly below `shape('turtle')`.[3] What is the result of this? The possible numbers are 0 to 10.

d) Change two of the numbers in the code so that the result becomes a rectangle with width 100 and length 200. See figure 1.8.

Figure 1.8: *Rectangle.*

e) Add code to draw figure 1.9.

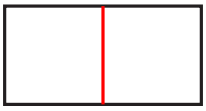


Figure 1.9: *Rectangle split in two halves.*

Use `color('red')` to color the center vertical line segment red.

[2] The terms *command*, *line* and *instruction* are used interchangeably.

[3] The weg page docs.python.org/3/library/turtle.html contains an overview of available commands in Python Turtle.

1.5 Drawing with Coordinates

118

Code

```
from turtle import *
shape('turtle')
penup()
goto(100, 0)
pendown()
goto(0, 100)
```

Tom understands the instructions `goto`, `penup()` and `pendown()`.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

When we want to test what happens if we remove a line of code, we can use a comment, like `#penup()`, instead of deleting the whole line.

c) Examine what happens if the line `penup()` is removed from the code. Do this by changing the line to `#penup()`.

d) Change `#penup()` back to `penup()`. Let `penup()` and `pendown()` switch lines. Remember to always guess the result, before you rerun the program.

e) Add the line `goto(-100, 0)` to the bottom of the code.

f) Complete the code to produce figure 1.10.

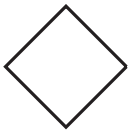


Figure 1.10: *A rhombus.*

g) Add more `goto`-commands to draw figure 1.11.

Figure 1.11: *A rhombus in a square.*

1.6 Circles and Fillcolors

Code

```
1 from turtle import *
2 shape('turtle')
3 fillcolor('red')
4 begin_fill()
5 circle(50)
6 end_fill()
```

Recipe

```
Create the turtle
Set the shape to turtle
Select red fillcolor
Begin the paint job
Draw a circle with radius 50
Finish the paint job
```

Tom can draw circles and fill figures with colors.

a) Read the code above, and then read the recipe. Guess the result.

b) Write the code, and run it. Change the fillcolor to light blue. That is, change line 3 to `fillcolor('lightblue')`.

c) Switch the number tallet 50 with other numbers like 120 and 300. Examine what happens if you try a negative number, for example −120.

d) Change the circle-command back to `circle(50)`, and add the following code on the bottom of the program:

```
fillcolor('yellow')
begin_fill()
circle(90)
end_fill()
```

Note: Did you remember to guess the result before you ran the program?

e) Let the colors `'yellow'` and `'lightblue'` switch lines.

f) Switch the numbers 50 and 90. What is the result of this?

g) Draw an eye similar to figure 1.12. Use the colors `'lightgray'`, `'lightblue'` and `'black'`.

Figure 1.12: *The all-seeing eye.*

1.7 Summary of Chapter 1

Python	English
`from turtle import *`	Import turtle-commands
`forward(150)`	Move 150 steps forward
`backward(40)`	Move 40 steps backwards
`left(60)`	Turn 60 degrees left
`right(70)`	Turn 70 degrees right
`circle(120)`	Circle with radius 120
`shape('turtle')`	Set the shape to a turtle
`penup()`	Pen up
`pendown()`	Pen down
`hideturtle()`	Hide the turtle
`goto(40, 80)`	Go to the point (40, 80)
`speed(8)`	Set the speed to 8
`color('yellow')`	Set yellow color
`fillcolor('blue')`	Set blue fillcolor
`begin_fill()`	Begin fillcolor
`end_fill()`	End fillcolor

Tabell 1.1: *Dictionary for chapter 1.*

To fill figures with a color, you must first set a fillcolor. For example: Write `fillcolor('brown')`, then follow these three steps:

1) Write the line `begin_fill()`.

2) Draw a closed figure. For example a circle, a triangle, or a rectangle.

3) End the code with the line `end_fill()`.

Note

Every program with Python Turtle needs to begin with `from turtle import *`.

1.8 Exercises

1.8.1 Correct Order

119


```
forward(200)
shape('turtle')
from turtle import *
```

```
forward(200)
left(45)
```

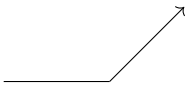


Figure 1.13: *An angle.*

Draw figure 1.13. Do this by ordering the lines above correctly.

1.8.2 Correct the Mistakes

119

Code

```
1 from turtle import *
2 shape(turtle)
3 Forward(300)
4 left[-180]
5 forwrad 300
```

Write the code above, and run the program. You will get at least one error message. Keep correcting the mistakes until the program runs without error messages.

1.8.3 Zorro

120

Figure 1.14: *Zorro. Fencing champion and nobleman.*

Write code to draw the Zorro-mark Z. Use the color `color('gold')`.
Write `pensize(8)` to increase the thickness of the lines.

1.8.4 Follow the Recipe

120

Recipe

```
Create the turtle
Go 150 steps forward
Turn 120 degrees left
Go 150 steps forward
Turn 120 degrees left
Go 150 steps forward
Turn 120 degrees left
Turn 60 degrees right
Draw a circle with radius 87
```

Follow the recipe above, and translate to Python-code. What is the resulting figure?

1.8.5 Parallelogram

120

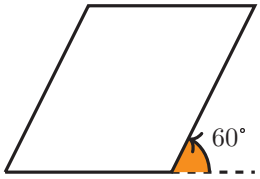


Figure 1.15: *Parallelogram.*

Draw a parallelogram. See figure 1.15. A parallelogram consists of two pairs of sides which are parallel and equal in length.

1.8.6 Two Squares and a Circle

Figure 1.16: *A circle between two squares.*

Write code to draw figure 1.16. Use the colors `'red'`, `'gold'` and `'black'`.

CHAPTER 2

Repetitions and Patterns

Figure 2.1: *A railroad: repeating pattern.*

2.1 Two repetitions

121

Code

```
from turtle import *
shape('turtle')

for i in range(2):
  circle(50)
forward(100)
```

The code above uses a for-loop. We use for-loops when we want Python to repeat instructions.

a) Read the code, and guess the result.

b) Write the code, and run the program. Notice that the command `circle(50)` is indented inward. Press either Tab once or Space twice, before you write `circle(50)`. Did you receive an error message when you ran the program? Check the list of usual mistakes on the next page.

Correct

```
for i in range(2):
  circle(50)
forward(100)
```

Wrong

```
for i in range(2):
circle(50)
forward(100)
```

Wrong

```
for i in range(2)
  circle(50)
forward(100)
```

You will now be asked to make some small changes to your program. Remember to always guess the result after each small change, before you rerun the program.

c) Add the line `speed(7)` directly below `shape('turtle')`.

d) Indent the command `forward(100)` inward like this:

```
for i in range(2):
  circle(50)
  forward(100)
```

e) Let the commands `circle(50)` and `forward(100)` swap places.

f) Add the command `circle(25)` in the beginning of the for-loop like this:

```
for i in range(2):
  circle(25)
  forward(100)
  circle(50)
```

g) Change the order of the lines inside the for-loop to produce figure 2.2.

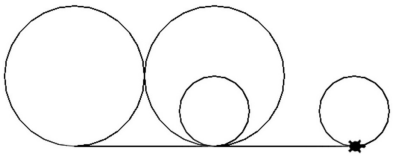


Figure 2.2: *Big and small circles.*

2.2 Three or More Repetitions

Code

```
1 from turtle import *
2 shape('turtle')
3
4 for i in range(3):
5   forward(150)
6   backward(150)
7   left(25)
```

Recipe

```
Create the turtle
Set the shape to a turtle

Repeat three times:
  Walk 150 steps forward
  Walk 150 steps backward
  Turn 25 degrees left
```

a) Read the code above, and guess the result.

b) Write the code, and run the program.

A gentle reminder: Always guess the result of each small change to the code.

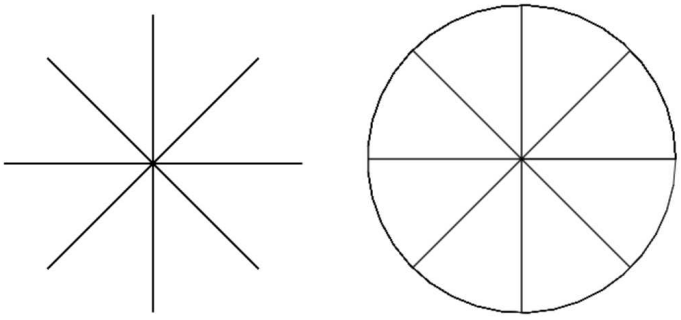
c) Add the line `speed(6)` above the for-loop.

d) Change `range(3)` to `range(6)`.

e) Change `left(25)` to `left(45)`.

f) Change the number of repetitions in order to produce figure 2.3a.

g) Add code below the for-loop to draw figure 2.3b. The commands `circle(150)`, `forward(150)` and `left(90)` may be useful.



(a) Without a circle. *(b) With a circle.*

Figure 2.3: *A pizza.*

2.3 Angles and Regular Polygons

121

Code

```
1 from turtle import *
2 shape('turtle')
3 speed(5)
4
5 for i in range(3):
6   forward(100)
7   left(120)
```

a) Read the code above, and guess the result.

The algorithm above can be illustrated with a *flowchart*. See figure 2.5 on page 17.

b) Write the code, and run the program.

c) Let lines 6 and 7 swap places. What happens?

d) Draw a square by changing the numbers 3 and 120.

Figure 2.4: *A hexagon.*

e) Change the numbers in the commands `left` and `range` to produce figure 2.4 (a hexagon).

Number of sides	Angle size
3	120
4	90
6	60

Tabell 2.1: *Number of sides versus angle size.*

f) Use the pattern shown in table 2.1 to draw a 18-sided polygon.[1] Increase the drawing speed by writing `speed(9)`. Set the side length to 50.

[1] Tip: The product is always 360°. For a rectangle we have $4 \cdot 90 = 360$ and for a hexagon $6 \cdot 60 = 360$, and so on.

Flowchart

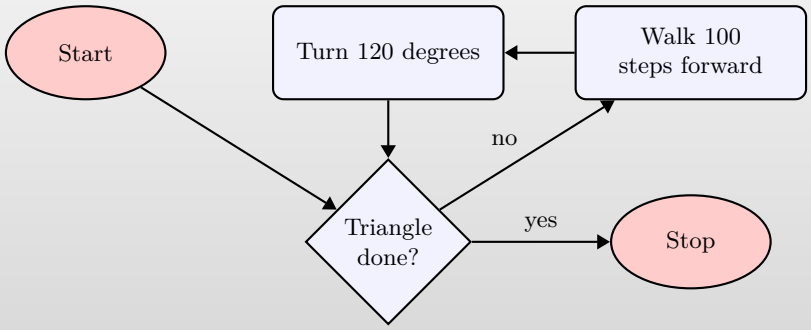


Figure 2.5: *Flowchart for a for-loop. Equilateral triangle.*

The use of a *flowchart* is a good way to visualize the flow of the code. It can improve our understanding of the code. A flowchart can also help us communicate the code that we have created to others.

2.4 Staircase Up and Staircase Down

122

Code

```
from turtle import *
shape('turtle')
speed(5)

for i in range(5):
  forward(25)
  right(60)
  forward(25)
  left(60)
```

The code above is our first attempt to draw a staircase.

a) Read the code above, and guess the result. Is the staircase moving upward or downward?

b) Write the code, and run the program.

c) Edit the code to produce 90-degree angles.

d) Change the for-loop such that it only repeats once. Do this by changing `range(5)` to `range(1)`. This will help you understand exactly what is being repeated.

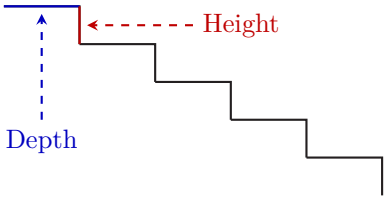


Figure 2.6: *Stairs down.*

e) Change back to `range(5)`. Write code such that the depth of a stair is twice that of the height of a stair. See figure 2.6.

f) Change the code such that the stairway is moving upward instead of downward.

Recipe

```
Draw staircase moving upward
Move a few steps forward
Draw staircase moving downward
```

g) Finish the code to draw figure 2.7. The recipe above shows how you can split this problem into three parts.

Figure 2.7: *Stairs up and stairs down.*

2.5 Experimenting with a For-loop

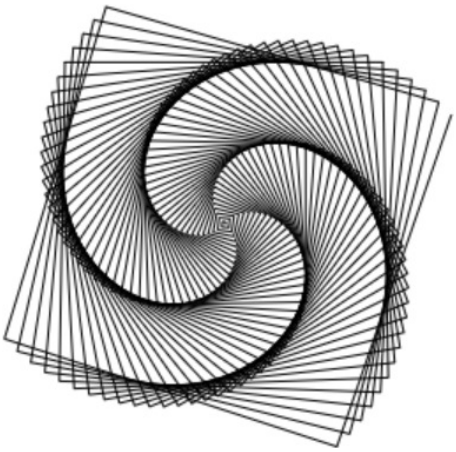


Figure 2.8: *Experimenting with a for-loop.*

Now you have your chance to experiment with a for-loop and see what kind of art you are able to create. When you change the command `forward(50)` to `forward(i)`, many things can happen. Figure 2.8 is an example of this. Use the code below as inspiration, write your own code and save your artwork as a desktop background.

Code

```
1 from turtle import *
2 shape('turtle')
3 speed(0)
4
5 for i in range(200):
6   forward(i)
7   left(89)
```

Recipe

```
Walk 0 steps forward
Turn 89 degrees left
Walk 1 step forward
Turn 89 degrees left
Walk 2 steps forward
Turn 89 degrees left
...
```

The two final instructions are “walk 199 steps forward” and “turn 89 degrees left”.

2.6 Summary of Chapter 2

Python	English
`for i in range(3):`	Repeat 3 times
`for i in range(25):`	Repeat 25 times

Tabell 2.2: *Dictionary for chapter 2.*

Here is an example of how we make a for-loop repeat something 6 times:

```
1 from turtle import *
2
3 for i in range(6):
4   circle(20)
5   forward(40)
6 circle(20)
```

The result of the code is figure 2.9.

Figure 2.9: *Seven circles.*

An overview of available commands for Python Turtle is listed on the web page `docs.python.org/3/library/turtle.html`.

Note

Only indented lines of code will be repeated in a for-loop. The two pieces of code below do exactly the same thing.

```
for i in range(2):
  forward(50)
  left(75)
forward(100)
```

```
forward(50)
left(75)
forward(50)
left(75)
forward(100)
```

2.7 Exercises

2.7.1 Draw a Square

122

Code

```
from turtle import *

for i range(4)
  forward('100')
left(90)
```

Write the code above, and run the program. Correct the mistakes such that the result is a square.

2.7.2 Draw a Rectangle with a For-loop

122

Figure 2.10: *Rectangle.*

Draw a rectangle with length 200 and width 100. See figure 2.10.

2.7.3 A Secret Figure

123

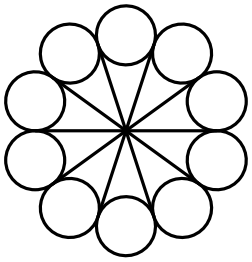
Recipe

```
Repeat 8 times:
  Turn 45 degrees left
  Walk 50 steps forward
  Turn 90 degrees right
  Walk 50 steps forward
```

Translate the recipe above to Python-code. What is the resulting figure?

2.7.4 Cones with Icecream

123



```
left(36)
hideturtle()
circle(26)
forward(80)
```

```
from turtle import *
speed(9)
for i in range(10):
backward(80)
```

Figure 2.11: *Icecream cones.*

Write a program which draws figure 2.11. Do this by writing the lines above in the correct order.

2.7.5 The Railroad

?

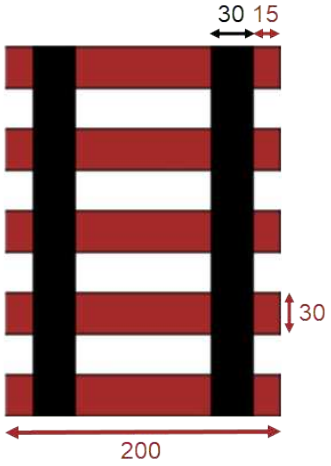


Figure 2.12: *A railroad with colors.*

Draw figure 2.12. In addition to for-loops, here are some commands you might find useful:

```
fillcolor('brown')
penup()
pendown()
begin_fill()
end_fill()
```

CHAPTER 3

Storing Code in Functions

3.1 My smile

123

Code

```
1  from turtle import *
2
3  # Set direction upward
4  setheading(90)
5  # Draw a half-circle
6  circle(50, 180)
```

Figure 3.1: *A smiling lady.*

The code above is an attempt to draw a smile.

a) Read the code above, and guess the result.

b) Write the code, and run the program. Can you explain the result?

c) Change the number 90 to other numbers like 0, 180, −45 or 270.

d) Change the number in the command `setheading` such that the program produces a smile.

When we are satisfied with our smile, we can store it as a function like this:

```
1  from turtle import *
2
3  # Store the function smile
4  def smile():
5    setheading(-90)  # Set heading downward (south)
6    circle(50, 180)  # Draw a half-circle with radius 50
7
8  # Smile once
9  smile()
10 # Smile once more
11 smile()
```

e) Write the code above, and run the program.

f) Add two new lines with the command `smil()`. Finish with the line `home()`. Guess the result, before you run the program.

```
1  from turtle import *
2
3  # Recipe of a smile
4  def smile():
5    setheading(-90)  # Set heading downwards (south)
6    circle(50, 180)  # Draw a half-circle with radius 50
7
8  def pout():
9    setheading(90)  # Set heading upward (north)
10   circle(50, 180)
11
12 smile()
13 pout()
14 pout()
15 pout()
```

We expand the program with a function to draw a pout.

g) Add the new code above to your program. Guess the result, then run the program.

h) Change the order of the instructions on line 12 to 15 to produce figure 3.2.

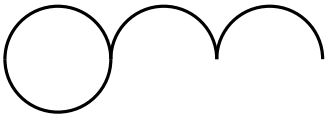


Figure 3.2: *A smile and pouts.*

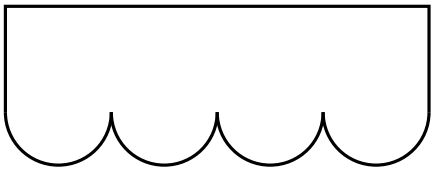


Figure 3.3: *An awning.*

i) Draw an awning as depicted in figure 3.3. Below are some commands which may be of use.

`forward`	`left`	`backward`
`right`	`smile()`	`pout()`

3.2 Cogwheel and Squares

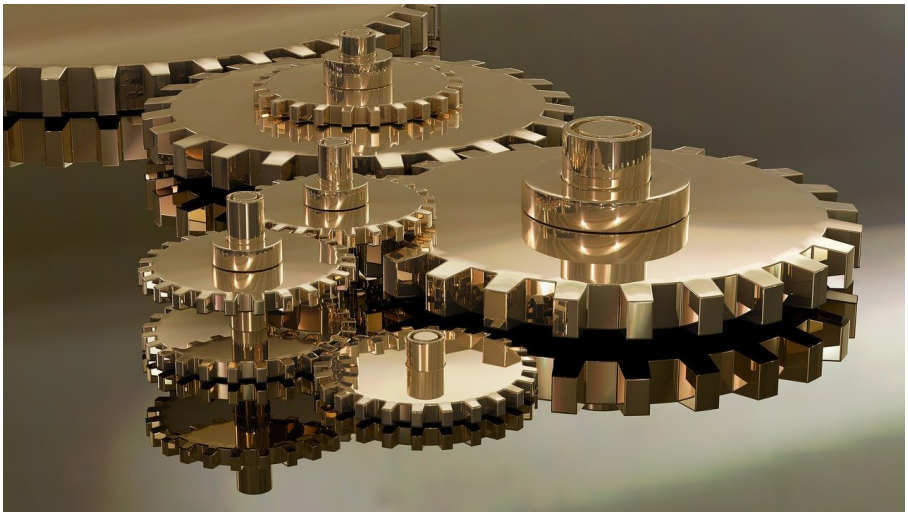


Figure 3.4: *A cogwheel. Used in a mechanical watch.*

We are going to create a program to draw a cogwheel.

a) Read the code below, and guess the result.

Code

```
1  from turtle import *
2
3  def square():
4    for i in range(4):
5      forward(50)
6      left(90)
7
8  square()
9  backward(100)
10 square()
```

Recipe

```
Draw a square
Walk 100 steps backward
Draw a square
```

b) Write the code, and run the program.

c) Let the lines 8 and 9 swap places. Run the program.

d) Use the code below as a starting point to draw figure 3.5. Change the order of the commands `square()`, `left(45)` and `forward(120)` to accomplish this.

Code

```
from turtle import *
speed(9)

def square():
  for i in range(4):
    forward(50)
    left(90)

square()
forward(120)
square()
left(45)
left(45)
square()
forward(120)
```

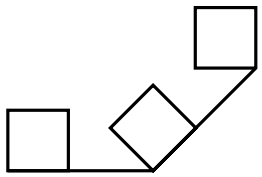
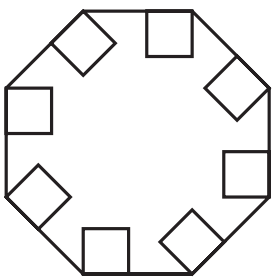
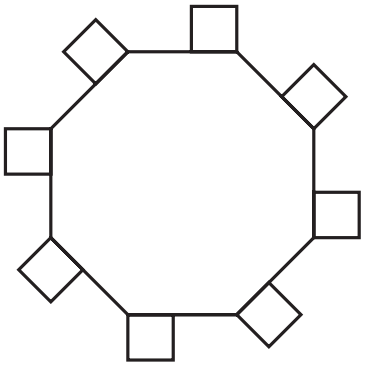


Figure 3.5: *A part of a cogwheel.*

e) Write code to complete the cogwheel. See figure 3.6a.
Tip: Consider which instructions are repeated. Use a for-loop to repeat these instructions.



(a) *Teeth facing inward.*



(b) *Teeth facing outward.*

Figure 3.6: *Cogwheels.*

f) Finally, change the code such that the cogwheel's teeth face outward. See figure 3.6b.

3.3 Varying the Size of Squares

Code

```
1  from turtle import *
2  speed(7)
3
4  def square(side):
5    for i in range(4):
6      forward(side)
7      left(90)
8
9  square(50)
10 square(60)
11 square(70)
```

Recipe

```
Draw a square with side 50
Draw a square with side 60
Draw a square with side 70
```

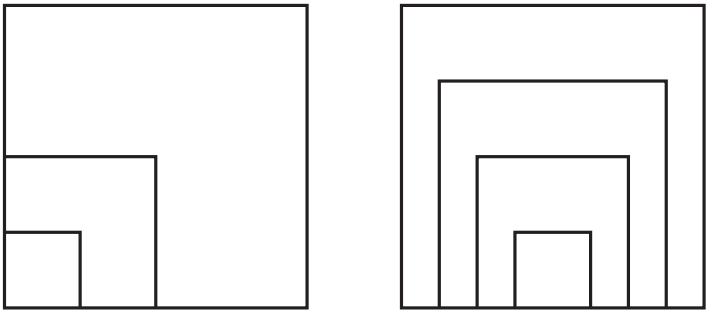
When we define a function such as `def square(side)`, we call `side` a *parameter*.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Add the commands `square(150)` and `square(-150)` at the bottom of the program. What happens?

d) Change the code to produce figure 3.7a, where the smallest square has side 50.



(a) Three squares. *(b) Four squares – growing.*

Figure 3.7: *Two square-patterns.*

e) Finish your code to produce the four squares in figure 3.7b. You may also need the commands `forward` or `backward`.

3.4 Tunnel in 3D

125

Code

```
from turtle import *
speed(0)

def square(side):
  for i in range(4):
    forward(side)
    left(90)

sidelength = 200  # Side length starts at 200
while True:
  square(sidelength)
  sidelength -= 10  # Decrement side length by 10
```

We create a *variable* for the side length, and we assign it the value 200 by writing `sidelength = 200`. To create a loop which repeats forever, we can write `while True:`.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Change the line `while True:` to `while sidelength > 50:`. This program can be visualized by the flowchart drawn in figure 3.8.

d) Change the line `sidelength -= 10` to `sidelength -= 50`. Guess the result before you run the program.

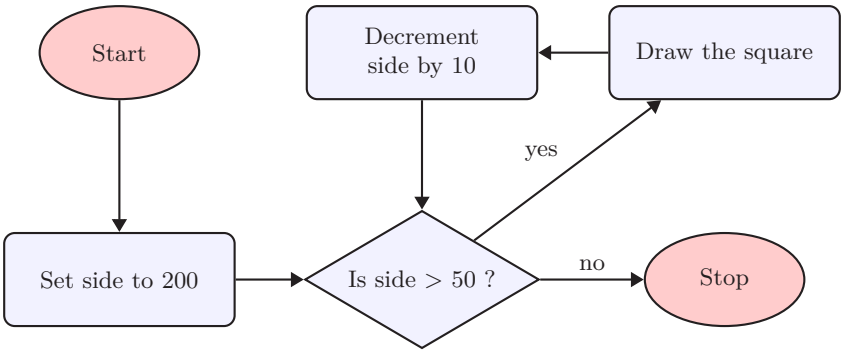


Figure 3.8: *Flowchart while-loop. Decreasing squares.*

Code

```
sidelength = 200  # The side length starts at 200
while sidelength > 10:
  square(sidelength)
  change = sidelength * 0.09
  sidelength -= change
```

The code above results in the variable `sidelength` decreasing by 9 % per repetition.[1]

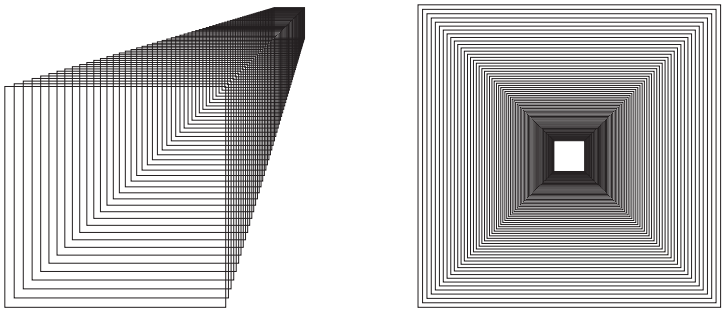
e) Change the while-loop as shown in the code above. Guess the result, before you run the program.

f) Change the number 0.09 to 0.25, and then to 0.03. Rerun the program after each change.

g) Add the following three lines of code inside the while-loop. Place them in the correct order to produce figure 3.9a.

`forward(change * 2)` `setheading(0)` `setheading(45)`

h) Change the command `forward(change * 2)` such that the result is figure 3.9b. Do this by replacing the number 2 with other numbers.



(a) Tunnel. A bird's-eye view. *(b) Tunnel. Ground perspective.*

Figure 3.9: *Tunnels.*

[1] To reduce the value of a variable by 9 %, you can also write `sidelength *= 0.91`.

3.5 Triangle-pattern with Midpoints

125

Code

```
from turtle import *
speed(9)

def triangle(side):
  for i in range(3):
    forward(side)
    left(120)

length = 300
triangle(length)
length /= 2  # Halve the length
left(60)
triangle(length)
```

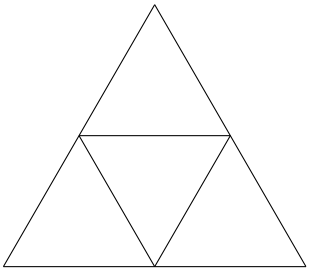


Figure 3.10: *Equilateral triangles.*

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Add the command `forward(length)` on the correct line to produce figure 3.10.

The triangle-pattern continues in such a way that the corners of a new triangle are the midpoints of the sides in the previous triangle.

d) Add code to draw the first three triangles.

e) Use a for-loop to draw the first seven triangles in this pattern. The result should look like figure 3.11.

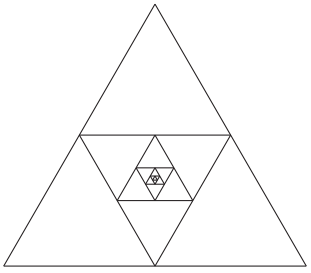


Figure 3.11: *Triangle-pattern with midpoints.*

3.6 Summary of Chapter 3

Python	English
`def`	Define a function
`def square():`	Define a function with the name square
`square()`	Run the square-function
`def square(side):`	Define a square-function with side as a parameter
`def square(side, color):`	Define a square-function with side and color as parameters
`circle(50, 90)`	Draw a quarter circle with radius 50
`circle(50, 180)`	Draw a half circle with radius 50
`while side > 10:`	Repeat as long as side is greater than 10

Tabell 3.1: *Dictionary of chapter 3.*

Here is an example of how to create a function with more than one parameter:

```
def smile(radius, my_color):
  color(my_color)
  setheading(-90)
  circle(radius, 180)

smile(100, 'blue')
```

Note

Only indented lines become part of the function.

```
def smile():
  setheading(-90)  # Inside of the function
circle(50, 180)  # Outside of the function
```

3.7 Exercises

3.7.1 A Function to Draw the Letter T

126

Code

```
from turtle import *
speed(1)  # Low speed to see each instruction better

def draw_T():
  forward(100)
  backward(30)
  right(70)
  forward(150)

# Draw T by calling on the function
draw_T()
```

Write the code above, and run the program. Change the code to produce the letter T.

3.7.2 Polygon-function

126

Code

```
from turtle import *
speed(8)

df polygon("edges")
  angle = 360 / edges
  for i in range(edges):
    forward(40)
    left(angle)

polygon(8)
```

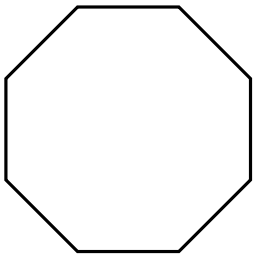


Figure 3.12: *An octagon.*

Write the code above, and run the program. Correct all mistakes, and finish the code in order to produce figure 3.12.

3.7.3 A heart

126

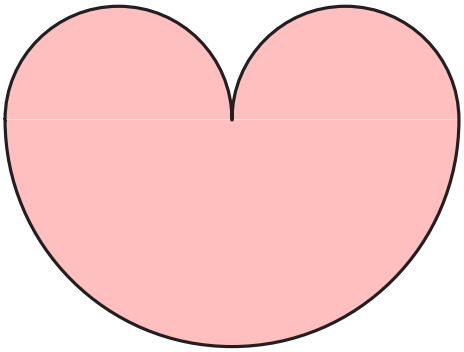


Figure 3.13: *A pink heart.*

Write a function which draws a pink heart. See figure 3.13. You may need the following lines of code:

`circle(100, 90)` `circle(50, 180)`

3.7.4 Eye Monster

127

Code

```
1  from turtle import *
2  speed(9)
3
4  def eyemonster():
5    circle(25)
6    fillcolor('gray')
7    begin_fill()
8    circle(15)
9    end_fill()
10   fillcolor('black')
11   begin_fill()
12   circle(10)
13   end_fill()
14
15 eyemonster()
```

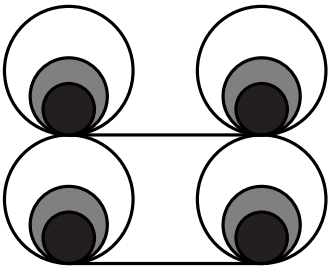


Figure 3.14: *A monster with four eyes.*

Use the code above as a starting point, then finish the code to draw figure 3.14.

3.7.5 Chessboard – The First Three Squares

127

Code

```
from turtle import *

# TODO: complete the
↪  function
def square(color):
  fillcolor(color)
  # Draw the square
  # Color the square

# Test the function
square('brown')
```

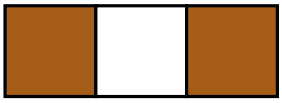


Figure 3.15: *Three squares.*

Write a program which draws the first three squares of a chessboard. Use the code above as a starting point. Set the side length of the square to 50. See figure 3.15.

3.7.6 Chessboard

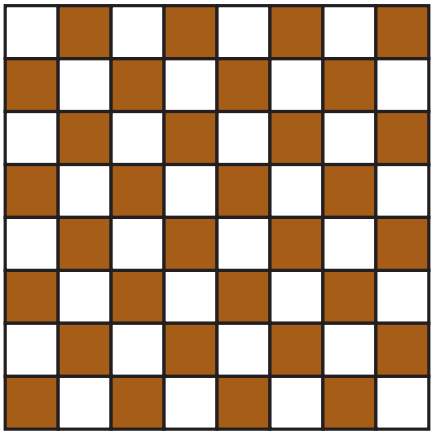


Figure 3.16: *A chessboard.*

Write a program to draw a chessboard similar to figure 3.16.

Tip: Begin with the first row. Create new functions if you need, for example `draw_even_row` and `draw_odd_row`.

CHAPTER 4
More Possibilities

4.1 A Wheel with Triangles

128

Code

```
1 from turtle import *
2 speed(9)
3
4 A = pos()
5 forward(80)
6 left(45)
7 forward(80)
8 goto(A)
```

Recipe

```
Store position as point A
Walk 80 steps forward
Turn 45 degrees left
Walk 80 steps forward
Go to point A
```

The command `A = pos()` stores the position of the turtle in the *variable* `A`.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Change the command `left(45)` by trying other numbers, for example 30, 140 and 270. Are you able to find several numbers which produce right-angled triangles?[1]

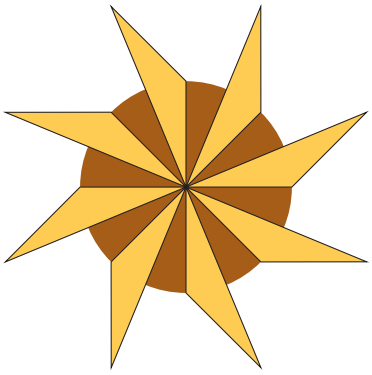


Figure 4.1: *A wheel with triangles.*

[1] A right-angled triangle: A triangle where one of the angles is 90°.

Some numbers in the command `left` will result in not getting a triangle. Can you find these numbers?

d) Add the line `side = 80` directly below `A = pos()`. Change both `forward`-commands to `forward(side)`. Run the program.

e) Change the value of the variable `side` by writing for example `side = 40` or `side = 150`. Try negative values, too. Do they work? Always guess the result, before you run the program.

f) Write a for-loop to repeat the drawing of the triangle several times. Begin like this:

```
side = 80
for i in range(3):
  A = pos()
  forward(side)
  left(45)
  # And so on
```

g) Change the command `range(3)` such that the triangles make a whole round.

h) Add the line `side = side * 0.9` at the bottom of the for-loop.[2] Change to 30 repetitions. To explore this further, try numbers other than 0.9.

i) Add the line `forward(side)` at the bottom of the for-loop. Guess how this will change the result. Run the program.

j) Draw figure 4.1. Here are some commands you may find useful:

`circle` `penup()` `fillcolor('brown')`
`pendown()` `begin_fill()` `fillcolor('gold')`
`end_fill()`

[2]The code `side = side * 0.9` can also be written like this: `side *= 0.9`.

4.2 Random Angles

Code

```
1  from turtle import *
2  from random import randint
3  speed(2)
4
5  angle_v = randint(10, 100)
6  print(angle_v)
7  forward(100)
8  backward(100)
9  left(angle_v)
10 forward(100)
```

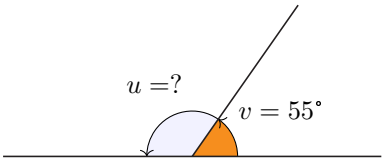


Figure 4.2: *Supplementary angles.*

The result of the command `randint(10, 100)` is a random integer from 10 to 100, both included.[3]

a) Read the code above, and guess the result.

b) Write the code, and run the program 5 times.

c) Change the command `randint(10, 100)` to `randint(90, 135)`.

d) Change the command `randint` to `randint(20, 130)`.

e) Add the following lines of code in the correct order to produce figure 4.2.

`print(angle_u)` `left(angle_u)`
`forward(100)` `backward(100)`
`angle_u = 180 - angle_v`

Run the program several times. Notice the output. It may look like this:

```
75
105
```

What is the relationship between these two angles? That is, what is their sum?[4]

[3] The word *randint* is short for "random integer".
[4] Euclid, Euclids Elements [2]

4.3 The Circumscribed Circle of a Triangle

129

Code

```
from turtle import *
from random import randint

shape('circle')
u = randint(0, 180)
print(u)
A = pos()

circle(100, u)
B = pos()
stamp()
```

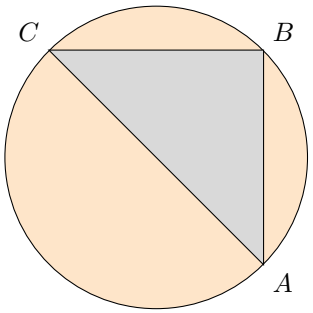


Figure 4.3: *The circumscribed circle.*

The command `circle(100, u)` draws an arc with radius 100 and angle u.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Create a new random angle v in the same way we created angle u. Use the `print`-command to see the value of the angle v.

d) Draw a new arc in the same way we drew the first arc, but the size of the angle is now v. Add the line `C = pos()` to store the position and add the line `stamp()` to mark the point.

e) Draw the final arc, such that you end up at the same point where the drawing began. *Tip:* The angle is $360 - u - v$.

f) Finish the code to draw the triangle inside the circle. See figure 4.3. The command `goto(B)` may be useful here.

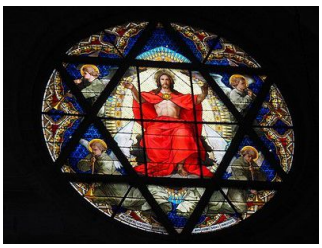


Figure 4.4: *The circumcircle in architecture.*

4.4 Bullseye

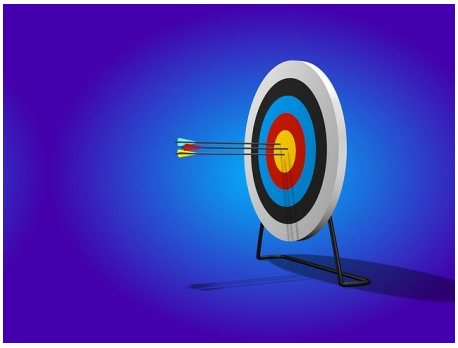


Figure 4.5: *Archery and bullseye.*

We are going to do draw a bullseye as shown in figure 4.5.

a) Read the code below, and guess the result.

Code

```
1 from turtle import *
2
3 radius = 50
4 increase = 25
5 circle(radius)
6 radius += increase
7 circle(radius)
```

b) Write the code, and run the program. Are we on the right track?

Between each circle we need to move a bit downwards. This distance is marked on figure 4.6.

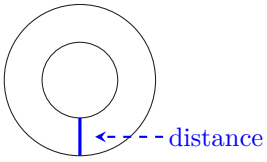


Figure 4.6: *Bullseye. Distance between circles.*

```
# Draw the first circle
distance = 15
right(90)
forward(distance)
left(90)
# Draw the second circle
```

```
# Draw the first circle
distance = 15
sety(ycor() - distance)
# Draw the second circle
```

Above you see two options to code this distance. The value of `distance` is not entirely correct, but we will correct this later.

c) Choose one of the two options above, and write the code. This code should be placed before you draw the second circle.

d) Expand the program such that it draws three circles. Change the command `avstand = 15` such that the distance between the circles looks about right.

e) Is it strictly necessary to use the variable `distance`? Could you have produced the same result by only using the variables `radius` and `increase`? Try it!

f) Use a for-loop to draw the five circles. Use the following code as a starting point:

```
for i in range(5):
  circle(radius)
  radius += increase
  # TODO: Lift the pen up
  # TODO: Move downwards
  # TODO: Set the pen down
```

g) Finish the code to color the bullseye as shown on figure 4.5. We see that the colors are yellow, red, blue, black and gray. We can change the for-loop such that it repeats itself for each of these colors. Use the following code as a starting point:

```
for color in ['yellow', 'red', 'blue', 'black', 'gray']:
  fillcolor(color)
  begin_fill()
  # TODO: Draw a circle
  end_fill()
  # TODO: Lift the pen up, and so on ...
```

4.5 Distance Alarm

130

Figure 4.7: *Little John exploring the outdoors..*

Maria has installed GPS-tracking on her son, Little John. The GPS-signals transmit to an app on her mobile phone. We will code a simplified simulation of how this might work.

a) Read the code below, and guess the result.

Code

```
1 from time import sleep
2 from turtle import *
3 from random import randint
4 shape('circle')
5 speed(1)
6
7 for i in range(5):
8   forward(30)
9   sleep(1)
```

Recipe

```
Sett shape to a circle
Set speed to 1
Repeat 5 times:
  Walk 30 steps forward
  Sleep 1 second
```

b) Write the code, and run the program.

c) Change `sleep(1)` to `sleep(2)`. Explore the `sleep`-command further by trying more numbers, such as 4 or 0.5.

d) Change `sleep` to `sleep(randint(0, 3))`. What happens?

Little John is not allowed to move farther from home than 140 steps. If this should occur, mom will receive a message on her mobile phone. We presume Maria will call on her son, and that he will always come home whenever she calls on him.

Code

```
print(distance(home))
if distance(home) > 140 :
  print('Come home!')
  home()
  break
```

Recipe

```
Print the distance home
If distance is greater than
↪  140:
  Print "Come home!"
  John walks home
  Cancel the for-loop
```

e) Write the code above, and place it at the end of the for-loop. Run the program.

Maria would like a mild warning whenever Little John is more than 80 steps away from home. To accomplish this, we write the following code:

```
if distance(home) > 140:
  print('Come home!')
  home()
  break
elif distance(home) > 80:
  print('Mild warning')
```

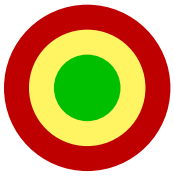


Figure 4.8: *Color zones.*

The word `elif` is short for "else if".

f) Read the code above, and guess the result.

g) Add the new code, and run the program.

h) Add the command `left(randint(0, 359))` directly above the `forward`-command. Guess the result, then run the program.

Little John usually walks more than just 5 steps. By writing `while True:`, we can create a loop which repeats forever.

i) Change `for i in range(5):` to `while True:`.

Maria wants the app to change color depending on how far away from home John is.

j) Finish the program such that the background color changes depending on Johns distance from home. You need to use the command `Screen().bgcolor('red')`, among others. See figure 4.8.

4.6 Summary of Chapter 4

Python	English
`print(distance(home))`	Print the distance from turtle to (0, 0)
`print('Come home!')`	Print the text “Come home!”
`home()`	Go home and set heading toward the right
`Screen().bgcolor('yellow')`	Set the background color to yellow
`stamp()`	Stamp the current shape of the turtle
`sleep(2)`	Sleep 2 seconds
`from time import sleep`	Required in order to use the command `sleep`
`randint(2, 4)`	Produce a random integer from 2 to 4
`from random import randint`	Required in order to use the command `randint`
`while True:`	A loop with infinite repetitions
`break`	Cancels a loop
`ycor()`	Turtle’s y-coordinate
`sety(50)`	Set the y-coordinate to 50
`pos()`	Give Turtle’s position
`radius = 50`	Store 50 in the variable radius
`radius += 30`	Increment radius by 30
`if`	if
`elif`	else if
`else`	else

Tabell 4.1: *Dictionary of chapter 4.*

Here is an example of how you can control the code with `if` and `else`:

```
if ycor() > 300:  # If y is greater than 300
  home()
elif ycor() >= 200: # Else if y is greater than or equal to 200
  forward(25)
else:  # Else: Here else means y less than 200
  forward(50)
```

4.7 Exercises

4.7.1 Scaling with a Variable

131

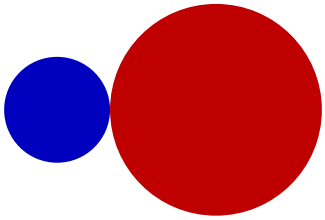


Figure 4.9: *Two colored circles.*

Write a program which draws figure 4.9. Do this by placing the commands below in the correct order.

```
forward(radius * 3)
```

```
from turtle import *
```

```
radius = 100
```

```
penup()
```

```
dot(radius, 'blue')
```

```
dot(radius * 2, 'brown')
```

4.7.2 Volume-symbol

131

Code

```
1  from turtle import *
2
3  width(5)
4  radius = 30
5
6  for i in range(8):
7    penup()
8    circle(radius, 30)
9    pendown()
10   circle(radius, 120)
11   penup()
12   circle(radius, -150)
13   # TODO: change the radius
14   sety(ycor() - 15)
```

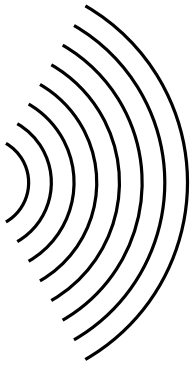


Figure 4.10: *Volume-symbol.*

Write the code, and run the program. Write code on line 13 such that the program produces figure 4.10.

4.7.3 One Circle and 100 Chords

131

Recipe

```
Set maximum drawing speed
Draw a circle with radius 100
Repeat 100 times:
  Draw an arc with a random angle (0-180)
  Store the position as A
  Draw an arc with a random angle (0-180)
  Store the position as B
  Go to point A
  Go to point B
```

A *chord* is a line segment between two points on a circle. Write a program which draws 100 chords in a circle. See figure 4.11. Use the recipe above as a starting point. You will also need the following commands:

```
circle(100, randint(0, 180))
from random import randint
B = pos()
goto(A)
```

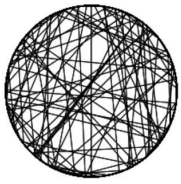


Figure 4.11: *100 chords.*

4.7.4 A watch

132

Figure 4.12: *A watch.*

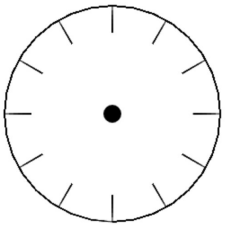


Figure 4.13: *A watch drawn with Python.*

Draw a watch similar to figure 4.13.

4.7.5 Treasure Hunting with a Compass

Code

```
from turtle import *
from random import randint
from time import sleep
shape('turtle')
penup()

# Place the treasure on a random spot
goto(randint(-200, 200), randint(-150, 150))
dot(20, 'gold')
treasure = pos()
home()
pendown()

# The treasure hunt begins
print(distance(treasure))  # Distance to the treasure
print(towards(treasure))  # Direction to the treasure

while distance(treasure) > 20:
  heading_treasure = towards(treasure)

  if heading_treasure > -45 and heading_treasure <= 45:
    setheading(0)  # Set direction east
  elif heading_treasure > 45 and heading_treasure <= 135:
    setheading(90)  # Set direction north
  # TODO: Add more directions
  forward(20)
  sleep(1)

print('Treasure found!')
```

Use the code above as a starting point. Finish the code such that the turtle always finds the treasure. The turtle is only allowed to move in one of the four directions north, south, east or west.

Figure 4.14: *A compass.*

CHAPTER 5

Mathematics in Python

5.1 Add, Subtract, Multiply and Divide

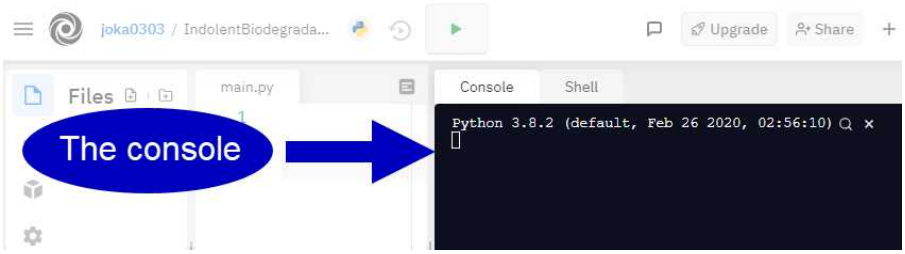


Figure 5.1: *The console on* `replit.com`.

We're going to have a live chat about mathematics with the computer. We will need to speak the language Python. We will call the computer by the name Jerry[1].

a) First, open the web page `replit.com/new/python3`.

The *console* is located in the black area of the screen. See figure 5.1.

b) Go to the *console*, and write your first message to Jerry. For example, try writing `Hello, Jerry`. Press Enter to send the message. Jerry will reply to you immediately.

```
Hello, Jerry
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Hello' is not defined
```

Notice that Jerry is telling us that "Hello" is not defined, and he calls the error a `NameError`.

[1] Jerry is inspired by the American science fiction thriller *Sphere*.

c) Write the message `'Hello, Jerry'` with the single quotation marks. What happens?

d) Ask Jerry to calculate $3+5$ by writing `3+5`. If you did this correctly, your console should look like this:

```
> 3+5
8
```

e) Write these three messages to Jerry: `3-5`, `3*5` and `3/5`. Remember to press Enter after each message. If you did this correctly, your console should look like this:

```
> 3-5
-2
> 3*5
15
> 3/5
0.6
```

Note that `3/5` betyr $3 \div 5 = 0.6$.

f) Press *clear* X on the top right of the console to delete the content.

g) Ask Jerry to complete the three calculations below.

$2 \cdot 3 - 6$	$10 \div 5 + 12$	$1/2 + 3/2$

After you are done, check your answers against the solution below.

```
> 2*3-6
0
> 10/5+12
14.0
> 1/2+3/2
2.0
```

5.2 Powers and Parentheses

```
> 2**2
4
> 2**3
8
> 2**4
16
```

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

Above you can see how we tell Jerry to calculate powers, such as 2^4.

a) Use the console (the black area), and write the following three messages to Jerry: `> 2**2`, `> 2**3` and `> 2**4`. You should get results equal to those shown above.

b) Use the console to calculate 3^4. You should get 81.

c) Write `> (3-5)**2+3` to calculate $(3-5)^2+3$. You should get 7.

d) Calculate $(1/5)^2$ by writing `> (1/5)**2`. This should yield the following result:

```
> (1/5)**2
0.04000000000000001
```

e) Calculate $(5-2\cdot 3)^4$. You should get 1.

5.3 Square Root

```
> from math import sqrt
> sqrt(4)
2
> sqrt(9)
3
```

$\sqrt{4} = 2$

$\sqrt{9} = 3$

Jerry can calculate square roots when we use the command `sqrt`[2]

a) Go to the console and write `> from math import sqrt`.

[2]The word *sqrt* is short for "square root".

b) Calculate $\sqrt{4}$ by writing `sqrt(4)`.

c) Calculate $\sqrt{9}$ by writing `sqrt(9)`.

d) Calculate the following:

$\sqrt{16}$	$5 - \sqrt{25}$
$\sqrt{10}$	$\sqrt{1/4}$

5.4 Some Algebra

```
a=3
a
3
a+2
5
```

The console above shows how we can store values in variables.

a) Write the three expressions (messages) `a=3`, `a` and `a+2`. The result should be as shown above.

b) Write `a=-2` and then `a+2`. Guess the result, before you press Enter.

c) Write `b=5` and then `a+b`. Guess the result, before you press Enter.

Note: If you clear the console by pressing *clear* , Jerry will forget the variables a and b.

d) Calculate $3b - 2a$ by writing `3*b-2*a`.

e) Calculate $3b - 2a$, but first set $a = 5$ and $b = -3$. You should get -19.

5.5 Variables and Rounding

Figure 5.2: *Red apples.*

Below you see an example of using variables in a practical problem.

Apples cost 1.10 $/lb.
How much does 2.7 lb of apples cost?

```
unitprice = 1.10
pounds = 2.7
price = unitprice * pounds
price
2.9700000000000006
```

a) Write the four messages above in the console.

b) Write `round(price)`. What happens?

c) Write `round(price,3)`. What happens? Try numbers other than 3.

Suppose the price the next day increases to 1.24 $/lb. Continue working in the same concole. Don't clear the console!

d) Now calculate the new price for 2.7 lb of apples. Write the following in the console. Remember to press Enter after each command.

```
unitprice = 1.24
price = unitprice * pounds
price
3.3480000000000003
```

e) Add the message `round(price,2)`. Guess the result, before you press Enter.

5.6 A Polite Greeting

132

Code

```
1 print('Hello! What is your name?')
2 name = input()  # Let the user type in his name
3 print('Hello, Mr. X')
```

Note: We have left the console (the black area) and are now writing the code in the white area again, just like we did in the first four chapters.

a) Read the code above, and guess the result.

b) Write the code on the web page `replit.com/new/python3`, and run the program.[3] When you run the program, it will wait for you to write your name in the console. Write your name, then press Enter.

c) Change `print('Hello, Mr. X')` to `print(f'Hello, Mr. {navn}')`.

Not everyone identifies with the title *Mister*. We will therefore let the user tell us his sex, in addition to his name.

d) Add the line `sex = input('What is your sex? (male/female): ')` directly below `navn = input()`.

If the user answers "Hanan" and then "female", then the output should look like this:

```
Hello! What is your name?
Hanan
What is your sex? (male/female): female
Hello, Ms. Hanan
```

e) Add the following lines of code such that the result of the program looks like the output above, assuming the user answered "Hanan" followed by "female":

`print(f'Hello, Ms. {navn}')` `if sex == 'female':`

f) Expand the program such that the output will be `Hello, Mr. Armstrong`, if the user answers "Armstrong" and then "male".

[3] We are now using *Python 3*, not *Python Turtle*.

5.7 Calculator Program

133

Code

```
1 numberA = input('Enter a number: ')
2 numberB = input('Enter a number: ')
3 thesum = numberA + numberB
4 print(f'{numberA} + {numberB} = {thesum}')
```

a) Read the code above, and guess the result. Note that we use the variable name `thesum`, because `sum` is a built-in Python-function and should therefore be avoided as a variable name.

b) Write the code on the web page `replit.com/new/python3`, and run the program.

To convert a string (text) to a whole number, we use the command `int`.[4]

c) Change lines 1 and 2 as shown below.

```
1 numberA = int(input('Enter a number: '))
2 numberB = int(input('Enter a number: '))
```

To subtract, multiply and divide we use `-`, `*` or `/` respectively.

d) Change the program such that it calculates the product. Hence, use `*` instead of `+`.

e) Rerun the program, and try to write in decimal numbers. For example 2.35 or 0.72.

We use the command `float` to convert to a decimal number.

f) Change the lines with input as shown below. Run the program, then enter a decimal number such as 3.59 or 10.234.

```
numberA = float(input('Enter a number: '))
numberB = float(input('Enter a number: '))
```

g) Add the line `quotient = numberA / numberB` to calculate the quotient. Run the program. Try to enter 0 as a value for `numberB`. What happens?

[4] The word *int* is short for "integer".

5.8 The Multiplication Master

133

Code

```
numberA = 3
numberB = 4
answer = numberA * numberB
print(f'{numberA} * {numberB} = {answer}')
```

a) Read the code above, and guess the result.

b) Write the code, and run the program.

Let's assign a random value to `numberA`.

c) Change the code as shown below.

```
from random import randint
# Generate a random integer between 0 and 10
numberA = randint(0, 10)
numberB = 4
answer = numberA * numberB
print(f'{numberA} * {numberB} = {answer}')
```

Run the program three times.

d) Change the last four lines in the code like this:

```
answer = numberA * numberB
print(f'{numberA} * {numberB} =')
user_answer = int(input('Your answer: '))
print(f'The correct answer is {answer}')
```

Run the program twice.

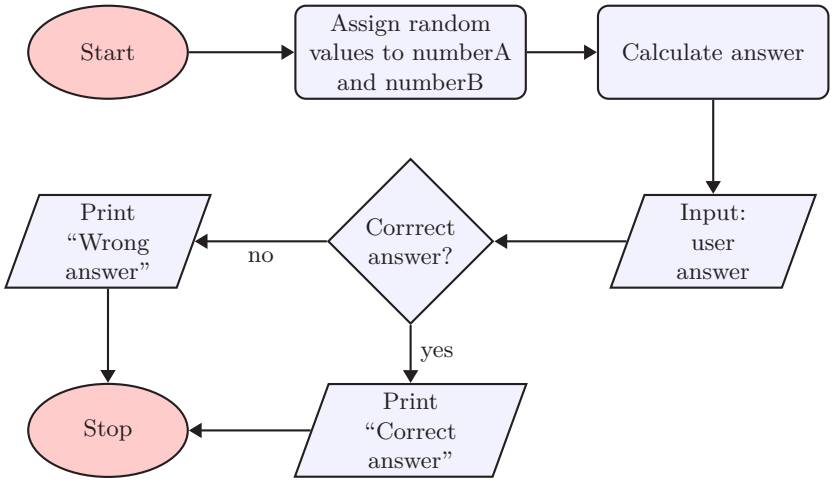


Figure 5.3: *Flowchart: random product.*

Let's check if the user provides a correct answer or not, and give feedback. A flowchart of the program we wish to code, is shown in figure 5.3.

e) Change the program as shown in the flowchart in figure 5.3. You may need the commands `if user_answer == answer:` and `else:` .

f) Create a for-loop which repeats the instructions in e) 10 times. Begin like this:

```
for i in range(10):
  numberA = randint(0, 10)
  # And so on ...
```

Remember: All code which belongs to the for-loop must be indented. If you forgot to indent the code, select all the code below the line `for i in range(10):`, and press Tab.

5.9 Summary of Chapter 5

Python	English
`round(price, 2)`	Round to 2 decimals
`text = input()`	Read input as a text
`number = int(input())`	Read input as a whole number
`desimal_number = float(input())`	Read input as a decimal number
`print('My program')`	Print the text “My program”
`2**3`	Calculate 2^3
`sqrt(1/4)`	Calculate $\sqrt{1/4}$.

Tabell 5.1: *Dictionary of chapter 5.*

The program will stop whenever it reads a line of code with `input()`. It will wait for the user to type a text or a number, and then to press Enter. All input is interpreted as a text (string). That's why we often need to convert input to either a whole number or a decimal number. See table 5.1. Use the command `print(f'...')` when you want to print text combined with the value from a variable. Here is an example:

```
temp = 15
print(f'The temperature is {temp} degrees.')
```

The result of the code is `The temperature is 15 degrees.`

Note

- To use the command `sqrt`, it's necessary that you first write the line `from math import sqrt`.
- When you need several math functions[5], for example `sqrt`, `sin`, `cos` or `log`, you can import all of them by writing `from math import *`.

[5]To see a complete list of available math functions in Python, see docs.python.org/3/library/math.html.

5.10 Exercises

5.10.1 The Average of Two Numbers

133

Code

```
1 print('Enter a number:')
2 number1 = float(input())
3 print('Enter another
  ↪  number:')
4 number2 = float(input())
```

```
Enter a number:
3
Enter another number:
5
The average is:
4.0
```

Write the code above, then add the following commmands in the correct order. The result should look like the output shown above right, if you enter the numbers 3 and 5.

```
print('The average is:')
```

```
print(average)
```

```
average = thesum / 2
```

```
thesum = number1 + number2
```

5.10.2 How Old Am I?

134

```
1 print('Enter your year of birth (e.g. 1982): ')
2 year = input()
3 age = year - 2021
4 print(f'You will be {age} years old this year.')
```

Write a program where the user can enter his year of birth. The output will be how old the user is this year. Use the code above as a starting point, and correct the mistakes.

5.10.3 Currency Converter

134

Write a program where the user can enter the amount in NOK (Norwegian kroner). The output will be the same amount, but in USD (American dollars). Use the following conversion rate:

$$1\,\text{USD} = 9.83\,\text{NOK}$$

To test your program, enter the number 983. The output should be 100 USD.

5.10.4 Temperature Check

134

Figure 5.4: *A thermometer.*

A thermometer, as shown in figure 5.4, measures the temperature. Write a program where the user can enter the temperature (in °C). If the temperature is below 0 °C, the program should print `Brr`. If the temperature is 0 °C or higher, the program should print `Getting warmer!`.

5.10.5 Map Scale

134

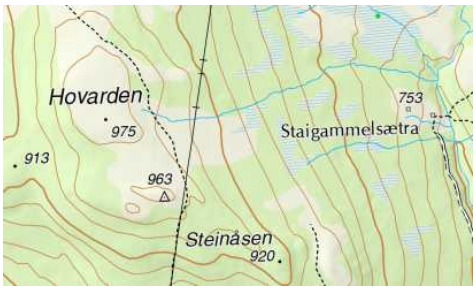


Figure 5.5: *A map of a small Norwegian town.*

Mustafa is an orienteer. He uses a map with the map scale 1 ÷ 20 000. This means that 1 inch on the map corresponds to 20 000 inches in the terrain. He runs with an average speed of 4.3 miles per hour.
Mustafa wants a program where he can enter the distance he measures on the map. The output should be both the distance in the terrain, and approximately how much time he's going to spend running. Here are some lines of code you may find useful:

```
speed = 4.3
distance_map_inches = float(input())
distance_terrain_inches = distance_map_inches * 20000
```

Tip: If the distance on the map is 5 inches, then the real distance in the terrain is 100 000 inches. To convert inches to miles, we calculate
100000 ÷ 63360 = 1.578 miles.

5.10.6 Recognizing Growth Factors

Code

```
from random import randint

# Random whole number between 0 and 100
percent = randint(0, 100)
growth_factor = 1 + percent/100
growth_factor = round(growth_factor, 2)  # Round to 2 decimals

print(f'Enter the growth factor of {percent} percent increase:')
answer = input()
answer = float(answer)  # Convert to decimal number
```

If the price of a product increases by 15 %, the growth factor is calculated like this:

$$\text{growth factor} = 1 + \frac{15}{100}$$

The code above shows the beginning of a program where a student can practice recognizing the relationship between the growth factor and the increase in percent. Finish writing the program and fulfill the following requirements:

- The user is given a random increase in percent. For example 27 percent increase.
- The user can enter the growth factor. For example he can enter 1.36.
- Upon answering correctly, the user will get the feedback `Correct!`.
- If the answer is wrong, the feedback should be in the form of `Wrong answer. The correct answer was 1.27`.
- The program shall continue to give new exercises until the end of the world. *Tip*: Use a loop to make this happen.

You may also find the following lines of code useful, to add a small pause between each exercise given to the user.

```
from time import sleep
```

```
sleep(1)
```

PART II

Applications in Mathematics

CHAPTER 6

Number Patterns

Learning Goals

- *Using programming to explore and produce number patterns.*
- *Using algorithmic thinking and programming to solve problems involving number patterns.*

6.1 Basic Number Patterns

135

Code

```
1 number = 0
2 for i in range(10):
3   print(number)
4   number += 2
```

Recipe

```
Assign 0 to number
Repeat 10 times:
  Print the value of number
  Increment number by 2
```

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python3`, and run the program.[1] Was the result as expected?

Remember to guess the result after each small change to the code, before you rerun the program.

c) Let `number` start on -4 insted of 0.

d) Change line 3 to `print(number, end=', ')`.

e) Change the number 10 to 50.

f) Change line 4 to `number -= 3`.

g) Let `number` start on 1 and change line 4 to `number *= 2`.

h) Change the code to produce the output `3, 6, 12, 24, ...`.

[1] You may also go to `replit.com`, and then create a new file (repl) with the language *Python*.

i) Change the code to produce the output `3, 6, 9, 12, ...`.

j) Change the code to produce the output `100, 95, 90, ...`. Also, change the number of repetitions such that 0 is the final number in the output.

k) Change the code to produce the output `100, 50, 25, 12.5, ...`.

6.2 A Fast-Growing Monkey Population

135

Code

```
1 monkeys = 4
2 # Double the number of
  ↪  monkeys
3 monkeys *= 2
4 print(monkeys)
```

Figure 6.1: *Two monkeys.*

A monkey population grows such that it doubles during the course of a year. We will examine how long it will take until the population reaches 5 million individuals.

a) Write the code above, and run the program.

b) Copy lines 3 and 4, and paste them at the bottom of the code. Run the program. In the console you should see the output `8` followed by `16`.

To repeat this growth process over many years, we declare a variable `year` which will increment by 1 for each year that passes.

```
1 monkeys = 4
2 year = 1
3
4 for i in range(10):
5   monkeys *= 2
6   print(monkeys)
```

c) Change the code as shown above. Run the program.

d) Add the line `print('Year Monkeys')` above the for-loop, and add the line `print(f'{year} \t {monkeys}')` inside the for-loop.[2]

e) Change the for-loop to correspond to a period of 30 years.

f) Add the line `year += 1` on the right place in the code, so that the years increase.

g) Add the following code inside the for-loop:

```
if monkeys >= 5000000:
  print('Above 5 million!')
  break  # Cancel the loop
```

h) Finish writing the code so that the program also prints out the number of years it takes until the population reaches 5 million.

6.3 Sums of Odd Numbers

135

Code

```
1 number = 1
2 thesum = 0
3 for i in range(5):
4   thesum += number
5   number += 2
6 print(thesum)
```

$$1 + 3 + 5 + 7 + 9 = 25$$

The code above calculates the sum of the first five odd numbers. Why do we use the variable name `thesum` and not `sum`? See the summary on page 72 for explanation.

a) Write the code above, and run the program.

b) Change the line `number = 1` to calculate $5+7+9+11+13$. You should get 45.

c) Add the line `print(number)` at the beginning of the for-loop. What happens?

d) Change the last line to `print(f'Sum = {thesum}')`.

e) Finally, change the code to calculate the sum $1+3+5+7+9+11+13+15$. Did you get 64?

[2] In Python, using `\t` gives a space similar to a press on Tab.

6.4 Sum of a Geometric Pattern

136

Code

```
# The first denominator is 2
denom = 2
for i in range(5):
  print(denom)
  # Double the denominator
  denom *= 2
```

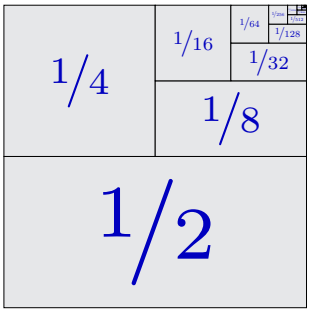


Figure 6.2: *A geometric pattern.*

Figure 6.2 shows a square with side length 1. The numbers on the figure represents the areas of the ever smaller rectangles. We notice that the denominators in the fractions form the number pattern $2, 4, 8, 16$ and so on.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

We will now create the number pattern $1/2, 1/4, 1/8, 1/16$ and so on.

```
denom = 2
for i in range(5):
  area = 1 / denom
  print(area)
  denom *= 2
```

c) Change the code as shown above, and run the program.

d) Use the for-loop to calculate $1/2 + 1/4 + 1/8 + 1/16 + 1/32$. Do this by adding the following lines of code and placing them correctly:

`print(thesum)` `thesum += area` `thesum = 0`

Let's examine what happens with the sum of the areas, if we include many more terms of the same number pattern.

e) Delete the line `print(area)`, and place the line `print(thesum)` inside the for-loop. Change the number of repetitions to 60. Finally, determine the sum: $1/2 + 1/4 + 1/8 + 1/16 + 1/32 + \ldots$.

6.5 Number Pattern with Circles

Figure 6.3: *Number patterns (fractals).*

The image above shows a beautiful mathematical image created with number patterns and programming.

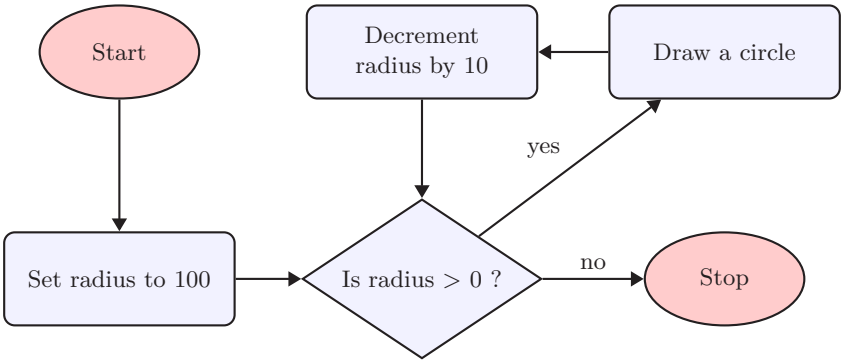


Figure 6.4: *Flowchart. A circle pattern.*

a) Read the flowchart in figure 6.4. Guess the result.

b) What number pattern is formed by the variable `radius` ?

Code

```
from turtle import *
speed(9)
radius = 100
while radius > 30:
  circle(radius)
  radius -= 10
```

c) Read the code above, and guess the result.

d) Write the code on the webpage `replit.com/new/python_turtle`, and run the program.

Remember to guess the result after each change, before you rerun the program.

e) Change `radius = 100` to `radius = 50`.

f) Change `while radius > 30:` to `while radius > 0:`.

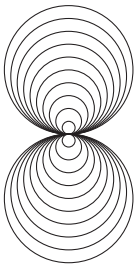
g) Change `radius -= 10` to `radius -= 5`

h) Change `while > 0:` so that the program draws circles with the following radiuses:

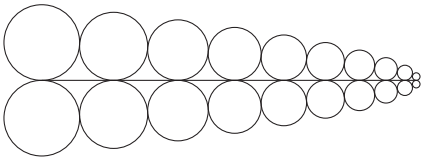
$$100, 90, 80, \ldots, -90, -100$$

You should get figure 6.5a.

i) Finish the code and create a program which produces figure 6.5b. You may need the commands `circle(-radius)` and `forward`.



(a) Circles with positive and negative radius.



(b) Circles forward.

Figure 6.5: *A circle pattern.*

6.6 The First 100 Natural Numbers

Code

```
from turtle import *

penup()
backward(200)
pendown()

for number in range(30):
  print(number)
  sety(number)
  sety(0)
  forward(5)
```

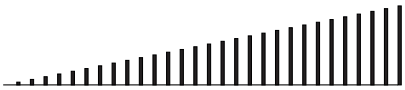


Figure 6.6: *Natural numbers.*

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python_turtle`, and run the program.

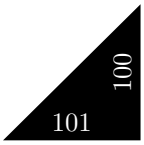
c) Change the `forward`-command to `forward(10)`. Guess the result before you run the program.

d) Change `sety(number)` to `sety(50)`. Guess how this will effect the height of the bars, then run the program.

e) Change `sety(50)` back to `sety(number)`.

f) Change `range(30)` in order for the final number to be 100. To increase the drawing speed, write the line `speed(9)` directly above `penup()`.

g) Change the `forward`-command such that the bars are displayed right next to each other. What kind of figure do you get?



$$1 + 2 + 3 + \ldots + 99 + 100$$

Figure 6.7: *The sum as a triangle.*

h) Calculate $1+2+3+\ldots+98+99+100$ by using the triangle in figure 6.7. *Tip:* Area of a triangle is $1/2 \cdot b \cdot h$, where b is the base and h is the height.

6.7 Rectangle Pattern

137

Code

```
from turtle import *
speed(8)
fillcolor('orange')

def rectangle(width, height):
  begin_fill()
  for i in range(2):
    forward(width)
    left(90)
    forward(height)
    left(90)
  end_fill()

# Draw a rectangle with width 10 and height 20
rectangle(10, 20)
```

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python_turtle`, and run the program.

c) Change the code to draw a rectangle with width 30 and height 90.

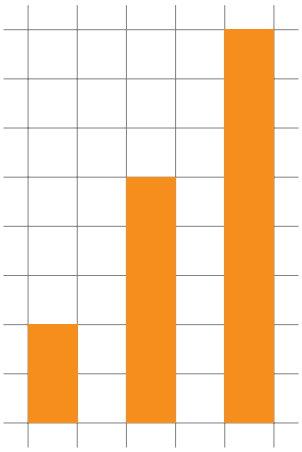


Figure 6.8: *A rectangle pattern.*

Recipe

```
Draw a rectangle with width 10 and height 20
Lift the pen up
Walk 20 steps forward
Put the pen down
And so on ...
```

d) Use the recipe above as a starting point, and then complete it so that the result of the recipe will be figure 6.8.

e) From the recipe you just created, write code to draw the three rectangles. The commands `penup()` and `pendown()` will be useful.

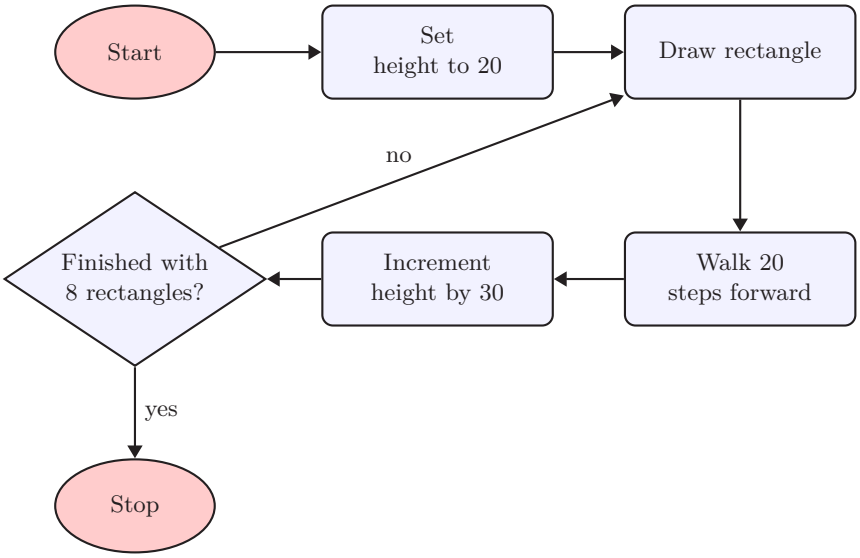


Figure 6.9: *Flowchart. A rectangle pattern.*

f) Using the pattern you analyzed in d) and e), write to code to draw the 8 first rectangles of the same pattern. See the flowchart in figure 6.9.

6.8 Summary of Chapter 6

Python	English
`print(number, end=',')`	Output ends with a comma
`print(number, end=' ')`	Output ends with a space
`print(number)`	Output ends with a linebreak
`print(f'{year} \t {monkeys}')`	\t produces a space equal to Tab
`while radius > 20:`	Loop repeats itself as long as radius > 20
`break`	Cancels a loop
`number *= 2`	Doubling of number
`number /= 2`	Halving of number
`number -= 5`	Decrement number by 5

Tabell 6.1: *Dictionary of chapter 6.*

We have seen how we can add numbers of a number pattern. For example, to add the first 50 numbers in the pattern $2 + 6 + 10 + 14 \ldots$ we write:

```
number = 2
thesum = 0
for i in range(50):
  thesum += number
  number += 4
print(thesum)
```

Note

We recommend avoiding the use of the variable name 'sum', because `sum` is already a built-in Python-function that is used to sum up the elements of a list.

6.9 Exercises

6.9.1 The 4 Times Table

137

Code

```
for number in range(0, 9):
  product = number + 4
  print(product)
```

```
4
8
12
...
36
40
```

Create a program which prints the the answers of the 4 times table. Use the code above as a starting point. Correct all the mistakes such that the output is equal to what is shown above next to the code.

6.9.2 Number Pattern with 100 Terms

137

Look at the following number pattern

$$10, 13, 16, 19, 22, \ldots$$

Here the first term is 10, the second term is 13 and so on. Create a program which prints the first 100 terms in this number pattern. Do this by placing the commands below in the correct order.

`number = 10`

`for term_nr in range(1, 101):`

`number += 3`

`print(term_nr, number)`

6.9.3 Dots in Circles

138

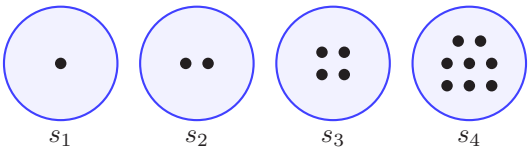


Figure 6.10: *Number pattern with figures.*

Figure 6.10 shows a number pattern with figures, where $s_1 = 1$, $s_2 = 2$ and so on. Create a program which prints the number of dots in each of the first 15 circles. Determine s_{15}, that is, what is the number of dots in circle number 15?

6.9.4 Number Pattern by Recipe

138

Recipe

```
Set number = 100000
Repeat as long as number >= 0.1:
  Print number
  Halve the value of number
```

Create a program which follows the algorithm of the recipe above. Then expand the program to decide the number of times you need to halve 100 000, until the value gets below 0.1.

6.9.5 Number of Terms in the Sum

138

The sum of the numbers in the 7 times table can be written as

$$7 + 14 + 21 + \ldots$$

Write a program to decide the number of terms we need to include such that the sum will be greater than 200 000.

6.9.6 Sum of an Infinite Number of Terms

138

A number pattern with an infinite number of terms is such that the first term is 1000, the second term is $1000 \cdot 0.9 = 900$, the third term is $900 \cdot 0.9 = 810$ and so on. The sum of these numbers can be expressed like this:

$$1000 + 900 + 810 + \ldots$$

Create a program which calculates this sum.

6.9.7 To Run a Marathon with Decreasing Speed

139

Figure 6.11: *A runner.*

Kilometer number	Speed in $^{km}/_{h}$	Time in hours
1	15,0	0,067
2	14,8	0,134
3	14,6	0,201
...	...	...

Tabell 6.2: *Speed and time data from the marathon.*

Stein, a Norwegian, is running a marathon. The length of one marathon is approximately 42 km (kilometers). He runs the first kilometer at speed 15.0 $^{km}/_{h}$ (kilometers per hour), the second kilometer at speed 14.8 $^{km}/_{h}$, the third kilometer at speed 14.6 $^{km}/_{h}$ and so on. See table 6.2.
Create a program to decide how long time Stein will spend to complete the marathon.
Tip: If he runs 1 km at the speed of 5 $^{km}/_{h}$, we calculate the time spent like this: $1 \div 5 = 0.2$. Hence, he uses 0.2 hours (12 minutes) on this kilometer.[3]

6.9.8 Saving under the Mattress and Buying Kebab

Fran follows a weird savings plan. The first three months he saves (and spends) his money in the following way:

- The first month he puts 1000 $ in his mattress, but he spends 1 % of these savings on kebab.
- The second month he puts 1100 $ in his mattress, but he spends 2 % of these savings on kebab.
- The third month he puts 1200 $ in his mattress, but he spends 3 % of these savings on kebab.

Fran continues to save his money in the same pattern. After some time, when he spends more than 100 % of the monthly savings on kebab, he needs to withdraw extra money from his mattress.

Create a program to examine how this savings plan will work out in the long run. What is the maximum amount of money he will ever have in his mattress? Will he ever spend all his savings? If yes, when will this happen?

[3] 0.2 hours in minutes is calculated like this: $0.2 \cdot 60 = 12$ minutes.

CHAPTER 7

Powers and Square Roots

Learning Goals

- *Programming with powers and square roots to explore patterns and solve problems.*
- *Using programming as a tool to better understand powers and square roots.*

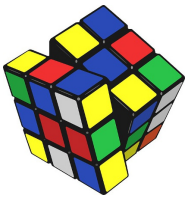


Figure 7.1: *Rubik's cube. The 3^3-edition.*

7.1 Practice Powers of 2

139

Code

```
from random import randint
print('Test yourself in powers of 2')
base = 2
exponent = 3
print(f'{base} ^ {exponent} = ')
answer = base ** exponent
print(f'Correct answer: {answer}')
```

A way of expressing 2 multiplied by itself 4 times, is $2^4 = \overbrace{2 \cdot 2 \cdot 2 \cdot 2}^{4} = 16$, where 2 is the *base* and 4 is the *exponent*.

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python3`, and run program.

c) Change `exponent = 3` to `exponent = 4`. Guess the result, before you run the program.

d) Change `exponent` to `exponent = randint(-2, 5)`. Guess the result, then run the program several times.

e) Change line 5 to `print(f'{base} ^ {exponent} = ', end='')`. Add the code `user_answer = float(input())` directly below. Run the program a few times. Remember that you need to enter a number in the console, and then press Enter.

```
if user_answer == answer:
  # Correct answer
  # TODO: Write feedback
else:
  # Wrong answer
  # TODO: Write feedback
```

Let us check whether the answer is right or wrong. The code above is a starting point for doing this.

f) Add the code above at the correct place in your program. You should replace lines starting with `# TODO` with a proper `print`-command. Run the program a couple of times.

g) Use `while True:` such that the program creates a new exercise after the user answers.

h) Finally, add code so that the program stops when the user has answered correctly 5 times. The following output should then be produced:

```
Congratulations! You answered correctly 5 times!
```

7.2 Explore Powers of 10

140

Code

```
print('Exploration of powers of 10.')  # A headline
print()  # Create a blank line
base = 10
exponent = 3
number = base ** exponent
answer_text = f'{base} ^ {exponent} = {number}'
print(answer_text)
```

Powers of 10 have base 10 and follow a specific pattern.

a) Read the code above, and guess the result.

b) Write the code, then run the program.

c) Change the line `exponent = 3` so that the program calculates 10^4. Now try the exponents 5 and 10. Do you see a pattern?

```
print('Exploration of powers of 10.')  # A headline
print()  # Create a blank line
base = 10
for exponent in range(2, 7):
  number = base ** exponent
  answer_text = f'{base} ^ {exponent} \t = \t {number}'
  print(answer_text)
```

Let us now use a for-loop to further explore this pattern. The code `\t` produces a space corresponding to the press of key Tab.

d) Read the code above, and guess the result.

e) Write the code, then run the program.

The result of the code `range(2, 7)` is that the loop repeats 5 times, but the values of the variable `exponent` vary from 2 to 6, both inclusive.

f) Change `range(2, 7)` so that the program calculates all powers of 10 from 10^1 to 10^9. Do you see a relationship between the exponent and the number of zeroes? Explain!

g) Let us now explore negative exponents. Change the `range` -command so that you calculate powers of 10 from 10^{-15} to 10^{-1}. Some of the answers look weird:[1]

```
10 ^ -5     =   1e-05
10 ^ -4     =   0.0001
10 ^ -3     =   0.001
```

h) Add the line `answer = f'{number:.10f}'` and then change the line with `answer_text` like this:

```
answer = base ** exponent
answer = f'{answer:.10f}'
answer_text = f'{base} ^ {exponent} \t = \t {answer}'
```

Run the program again. How many decimals are now present in the answers?

i) Change the number 10 in `answer = f'{answer:.10f}'` so that the first line of the output becomes:

```
10 ^ -15     =   0.000000000000001
```

j) Finally, it is possible to prettify the output by calculating the number of decimals from the exponent. Add the line `decimals = exponent * (-1)`, then change the line you added in h) to `answer = f'{answer:.{decimals}f}'`.[2] You should get the following output:

```
10 ^ -15     =   0.000000000000001
10 ^ -14     =   0.00000000000001
10 ^ -13     =   0.0000000000001
10 ^ -12     =   0.000000000001
10 ^ -11     =   0.00000000001
10 ^ -10     =   0.0000000001
10 ^ -9      =   0.000000001
10 ^ -8      =   0.00000001
10 ^ -7      =   0.0000001
...
```

k) Do you see a correlation between the negative exponent and the answer? Explain!

[1] The notation `1e-05` is called "scientific notation" and means 10^{-5}.

[2] When the exponent is -4, we get $10^{-4} = 0,0001$. Hence, we need $-4 \cdot (-1) = 4$ decimals.

7.3 Guess the Square Root

140

Recipe

```
Create a random number
Let answer be the square root of this number
Ask the user to guess the square root
Output the correct answer
```

Above you see a recipe for a program where the user can guess the square root of different numbers.

a) Write the code below, and run the program a few times.

```
from random import randint
from math import sqrt
number = randint(1, 101)
answer = sqrt(number)
print(f'Guess the square root of {number}: ')
guess = float(input())
print(f'You guessed {guess}. Correct answer: {answer}.')
```

b) Use the code below as a starting point. Finish writing it so that the user will know whether he guessed correctly, too high or too low.

```
if guess < answer:
  # TODO: Print output
elif guess > answer:
  # TODO: Print output
else:
  # TODO: Print output
```

When we calculate the square root of a number, we frequently get an answer with many deciamls, like 6.234456341234. It is unrealistic to expect the user to answer with such precision. We would like to accept the answer 6.2.

c) Add the command `answer = round(answer, 1)` directly below the line `answer = sqrt(number)` to round off the answer to one decimal.

d) Use a for-loop so that the user can continue guessing until the user guesses correctly. After each attempt, the program should tell the user whether the guess was correct, too high or too low.

7.4 Rice and Chess

140

Figure 7.2: *Rice on a chessboard.*

On a chessboard, We put one grain of rice on the first square, two grains on the second square and four grains on the third square. We continue adding grains of rice in the same pattern, as shown in figure 7.2.

a) Read the code below, and guess the result.

```
1 for square_nr in range(8):
2   print(sqaure_nr)
```

b) Write the code, and run the program.

We can write the numbers $1, 2, 4, \ldots$ as powers of 2: $2^0, 2^1, 2^2$ and so on.

c) Add the line `rice = 2 ** square_nr` at the correct place in the code. Change the `print`-command so that you output the number of grains of rice. Do you get the numbers $1, 2, 4, \ldots, 128$?

Let us now calculate the sum of all the grains of rice on the chessboard. The sum looks like this:

$$1 + 2 + 4 + 8 + \ldots = 2^0 + 2^1 + 2^2 + 2^3 + \ldots$$

d) Change `range(8)` so that you get the correct number of repetitions to include all 64 squares. Then add the following lines in the correct places:

`thesum+= rice` `print(thesum)` `thesum= 0`

Did you get the sum 18 446 744 073 709 551 615?

7.5 Summary of chapter 7

Python	English
`print(2**4)`	Outputs the answer of 2^4
`print()`	Creates a blank line
`for i in range(2, 7) :`	`i` is given the values $2, 3, 4, 5$ and 6
`print(f'{answer:.{3}}')`	Outputs `answer` rounded to 3 decimals
`answer = round(answer, 1)`	Rounds to 1 decimal
`number = randint(1, 101)`	Gives a random integer from 1 to 101

Tabell 7.1: *Dictionary of chapter 7.*

We have seen how we can create a practice program to practice math subjects such as powers and square roots. We use a loop to produce new exercises. Each new exercise can be created randomly by using the command `randint` or similar. Here is an example of writing user feedback:

```
if guess < answer:
  print('You guessed too low.')
elif guess > answer:
  print('You guessed too high.')
else:
  print('You guessed correctly!')
```

Note

- We calculate the square root with the command `sqrt`. This requires you to first import the command by writing `from math import sqrt`.
- To round off in the output, you can use the command `print(f'...')` like this:

```
answer = 0.12345566456
print(f'{answer:.{2}}')
print(f'{answer:.{4}}')
```

```
0.12
0.1235
```

7.6 Exercises

7.6.1 Square Numbers

141

Place the following lines of code correctly to create a program which outputs the first 20 square numbers, that is the numbers $1^2, 2^2, 3^2, \ldots, 20^2$.

```
number = base ** 2
```

```
for base in range(1, 21):
```

```
print(number)
```

7.6.2 Cubic Numbers

141

Code

```
term_nr = 1
cubic = term_nr ** 3
while cubic < 1000000:
  # TODO: Print the cubic number
  # TODO: Increment term_nr by 1
  # TODO: Calculate the next cubic number
```

The code above is a starting point to print the cubic numbers $1^3, 2^3, 3^3, \ldots$. Once the cubic number hits or exceeds 1 million (1 000 000), the code should stop printing cubic numbers. Complete the code. Add a `print`-command below the loop to print the first term number which produces a cubic number of 1 million or greater.

7.6.3 Danish Dan's DanPark

Danish Dan is a manager for the parking company DanPark. Customers who park their car at DanPark must pay 5 $ per hour. If the car is parked for at least 24 hours, the total price is calculated using the formula

$$\text{hours} \cdot 5 - \sqrt{\text{hours}^3}$$

Hence, if a customer is parked for 26 hours, the price is found by computing

$$26 \cdot 5 - \sqrt{26^3}$$

During the course of a particular week, 80 cars were parked at DanPark. The first car was parked for 1 hour, the second car for 2 hours, the third car for 3 hours and so on.

Create a program which outputs the price that each of the 80 customers needed to pay. Also, calculate the total income for Danish Dan's DanPark that week.

CHAPTER 8

The Coordinate System

Learning Goals

- *Using visual programming as a tool to increase understanding of the coordinate system.*
- *Using visual programming as a tool to distinguish between x- and y-coordinates, and explore and solve problems involving the coordinate system.*

This chapter also contains important prerequisite knowlegde to the subject of functions.

8.1 The Coordinate System – a Small Overview

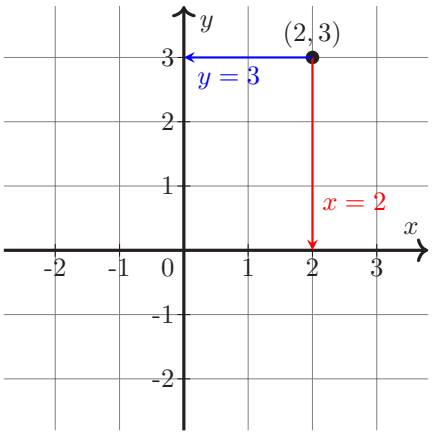


Figure 8.1: *A coordinate system with x- and y-axis.*

Above you see a coordinate system. The point $(2, 3)$ has x-coordinate 2 and y-coordinate 3. In other words, the point $(2, 3)$ means that $x = 2$ and $y = 3$, as shown in figure 8.1.

8.2 Explore x-coordinates with Python Turtle

141

Code

```
1  from turtle import *
2  shape('arrow')
3
4  def draw_xaxis():
5    circle(10)  # Draw a point where x = 0
6    setx(-220)
7    setx(220)
8    stamp()  # Place the arrow for the x-axis
9
10 draw_xaxis()  # Run the lines 5 through 8
```

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python_turtle`, and run the program.

c) Add the instruction `setx(-100)` on the line below `draw_xaxis()`.

```
for i in range(3):
  print('Enter a x-coordinate (-200 to 200):')
  x_coord = int(input())
  setx(x_coord)
```

d) Write the code above, at the bottom of your program. Run the program, and enter numbers such as 150, -100 or 0.

e) Add code such that the program draws a circle with radius 10, after turtle moves to the x-coordinate the user entered. If x is negative, make the circle red. If x is positive, make the circle blue. See figure 8.2. You may find the collowing lines of code useful:

`dot(10, 'red')` `dot(10, 'blue')`
`if x_coord < 0:` `else:`

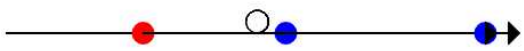


Figure 8.2: *Three x-coordinates. Positive and negative.*

8.3 Explore *y*-coordinates with Python Turtle

Code

```
1 from turtle import *
2 shape('arrow')
3
4 def draw_yaxis():
5   sety(-120)
6   sety(150)
7   stamp()
8
9 draw_yaxis()
```

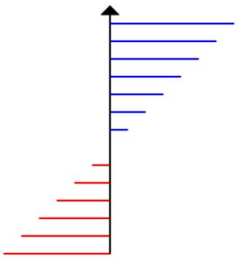


Figure 8.3: *y-axis. Red and blue.*

a) Read the code above. Guess how the *y*-axis will look like.

b) Write the code, and run the program.

c) Add either `left` or `right` with the correct angle, such that the *y*-axis looks correct. Add this command inside the function `draw_yaxis`.

d) Add the line `sety(0)` below the line `draw_yaxis()`. This is how we move the arrow back to the origin.[1]

Recipe

```
Set the y-coordinate to -120
Set the direction to the right
Repeat 6 times:
  Store the y-coordinate
  Walk y steps forward
  Walk y steps backward
  Increment the y-coordinate with 20
```

e) Translate the recipe above to Python code, and write this code at the bottom of your program. *Tip:* To store the *y*-coordinate, write `y = ycor()`. To increment y by 20, write `sety(y + 20)`.

f) Change the code accordingly to produce figure 8.3. Here are some lines of code which may be useful:

`color('black')` `color('red')` `color('blue')`
`if y > 0:` `elif y < 0:` `else:`

[1] The origin is the point where both x and y is 0, i.e. the point $(0, 0)$.

8.4 Parallel Lines

142

Code

```
1 from turtle import *
2
3 setx(200)
4 sety(50)
5 setx(0)
6 sety(100)
7 setx(200)
```

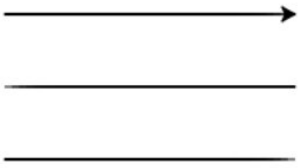


Figure 8.4: *Parallel lines. Horizontal.*

a) Read the code above, and guess the result.

b) Write the code, then run the program.

c) Change the program to produce three parallel lines as shown in figure 8.4. You will need the commands `penup()` and `pendown()`.

Recipe

```
Pen up
Set y to -150
Repeate 10 times:
  Pen down
  Set x to 200
  Set x to 0
  Pen up
  Increment the y-coordinate
  ↪  with 30
```

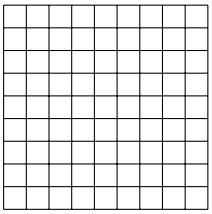


Figure 8.5: *9 x 9 grid.*

d) Change the program such that you draw 10 parallel lines. See the recipe on the above left. Here are some lines of code you may find useful:

`penup()` `pendown()`

`for i in range(10):` `sety(ycor() + 30)`

e) Add code to draw 10 vertical parallel lines, to produce the resulting grid. See figure 8.5.

8.5 Summary of Chapter 8

Python	English
`sety(100)`	Set the y-coordinate to 100
`setx(-50)`	Set the x-coordinate to -50
`print(xcor())`	Output the x-coordinate
`setx(xcor() + 30)`	Increment the x-coordinate by 30
`sety(ycor() - 20)`	Decrement the y-coordinate by 20
`shape('arrow')`	Set the shape on turtle to an arrow

Tabell 8.1: *Dictionary of chapter 8.*

In this chapter we used Python Turtle to explore x- and y-coordinates. When we write for example `setx(50)`, we move the turtle to a position where $x = 50$, without changing the y-coordinate. This works similarly for `sety(40)`. Here is an example:

```
1  from turtle import *
2
3  color('blue')
4  setx(50)
5  sety(-50)
6  setx(0)
7  color('red')
8  sety(ycor() + 100)
9  setx(xcor() - 50)
10 sety(0)
11 home()
```

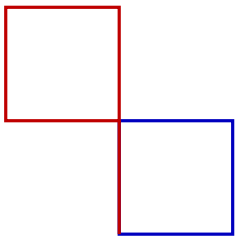


Figure 8.6: *Red and blue squares.*

Note

- The commands in this chapter are only available on the webpage `replit.com/new/python_turtle`.
- The line `from turtle import *` must be the first line of your program, for the turtle-commands to work.

8.6 Exercises

8.6.1 The Letter T – Again

143

Code

```
1 from turtle import *
2
3 setx(45)
4 sety(100)
5 setx(-45)
6 hideturtle()
```

Write the code above. Place the lines of code in the correct order to produce the letter T.

8.6.2 An Unknown Geometric Figure

143

Recipe

```
Read x from the user
Read y from the user
Go to the point (x, y)
Set y-coordinaten to -y
Go to the point (0, -2 * y)
Go back to the origin, i.e. the point (0, 0)
```

Translate the recipe above to Python code. You may find the following lines of code useful:

```
y = float(input())
```

```
print('Enter the y-coordinate:')
```

```
goto(0, -2 * y)
```

```
sety(-y)
```

What is the resulting geometric figure?

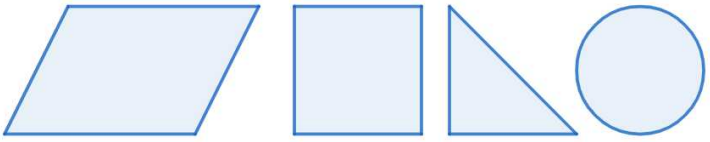


Figure 8.7: *Geometric figures.*

8.6.3 The Distance to a Point

143

Create a program where the user can enter a point by entering the x- and y-coordinates. The program should draw a line to this point. Then the program should calculate and print the distance d to this point (from the origin) by using the formula below. See figure 8.8.

$$d = \sqrt{x^2 + y^2}$$

In this exercise you may need the command `sqrt`. This command is imported by first writing `from math import sqrt`.

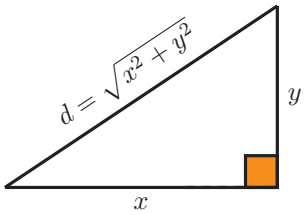


Figure 8.8: *The distance to a point from the origin.*

8.6.4 Random Points

Create a program which produces a random point, where x varies between -200 and 200, and y varies between -100 and 100. If both x and y are positive, then draw with a blue color. Else, draw with a red color. On the point itself, draw a small circle. Repeat all this 100 times. The result should look similar to figure 8.9. You must only use the commands `setx` and `sety` to move the turtle. You are not supposed to use `goto` in this exercise.

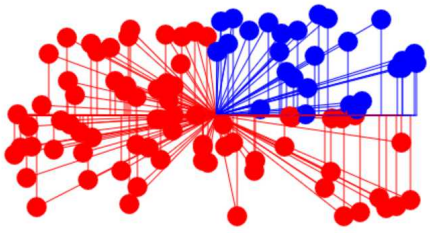


Figure 8.9: *Random points. Red and blue.*

CHAPTER 9

Functions

Learning Goals

- *To explore mathematical functions and their patterns using programming.*
- *To explore common relationships between variables found in the real world using programming.*
- *To program simple algorithms to find a specific function value.*

9.1 Kaare on the Café – without a Function

144

Code

```
1 price_per_cup = 4
2 cups = 3
3 price = price_per_cup * cups
4 print(f'Price: {price}')
```

Figure 9.1: *A cup of coffee.*

Kaare goes to a café to drink coffee. The price of a cup of coffee is 4 $. The code above calculates the price he needs to pay, when he buys three cups of coffee.

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/Python3`, then run the program.

One day Kaare bought six cups of coffee.

c) Change the code such that it calculates the price of six cups.

d) Change the code such that the user can enter the number of cups in the console, when the program is running. You will need the command `input`.

9.2 Kaare on the Café – with a Function

144

Code

```
1  def calculate_price(cups):
2    return cups * price_per_cup
3
4  price_per_cup = 4
5  price = calculate_price(3)  # The price for 3 cups
6  print(price)
```

We create a function by using the Python word `def`. In the code above, we call `cups` an *argument* or a *parameter*.

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/Python3`, and run the program.

c) Change the number 3 in `price = calculate_price(3)` to 5. Guess the result, before you run the program.

d) Add code to output a text such as:

```
5 cups of coffe cost 20 dollars.
```

e) Read the code below, and guess the result. Replace all lines below line 4 with this code, then run the program.

```
for my_cups in range(1, 10):
  price = calculate_price(my_cups)
  print(f'{my_cups} cups cost {price} dollars.')
```

f) Delete the `print`-command inside the for-loop. Add the following lines and place them correctly in the code.

`print('Kopper\t Pris i kr')` `print(f'{kopper}\t {pris} ')`

g) Add the code `\t` several more times until you are satisfied with how the table looks like. Finally, change `range(1, 10)` such that the final line in the output shows what 80 cups of coffee cost.

9.3 Price per Workout

Code

```
def y(x):
  return 30 / x

workouts = 2  # Two workouts
price = y(workouts)
print(f'Price per workout: {price} dollars.')
```

Figure 9.2: *Walter is working out.*

Walter pays 30 $ monthly as a member of a training facility. The price per workout, y, is given by the function

$$y = \frac{30}{x}$$

where x is the number of workouts during a month.

a) Read the code above, and guess the result.

b) Read the code, and run the program.

c) Change the code such that you calculate the price per workout, if the number of workouts is 12.

The price per month now increases to 50 $.

d) Change the program to accomodate for the new price. Set the number of workouts to 2, and you should get the result 25 $ per workout.

e) Examine what happens with the price per workout when you double the amount of workouts. Here is a recipe to do this:

```
Print a headline "Number of workouts, Price per workout"
Let workouts start at 1
Repeat 5 times:
  Calculate price per workout
  Print the number of workouts and the price per workout
  Double the amount of workouts
```

Translate the recipe to Python. To double the amount of workouts, you can write `workouts *= 2`. What is the relationship between the number of workouts and the price per workout?

9.4 Seats in a Cinema

145

Figure 9.3: *A young lady in a cinema.*

The number of seats in a cinema on a specific row is decided by the following pattern:

Row number	Number of seats
1	3
2	7
3	11

Tabell 9.1: *Row number vs seats.*

a) Read the code below, and guess the result.

Code

```
def seats(row_nr):
  return 4*row_nr - 1

# Prints seats on row number 1
print(seats(1))
```

b) Write the code, and run the program.

c) Add code to print the number of seats on row number 5.

d) Select the correct of these lines of code, such that you print how many more seats you find on row 11 compared to row 10.

```
print(seats(10) - seats(11))
print(seats(11) - seats(10))
print(seats(11 - 10))
print(seats(10 - 11))
```

Sanna is sitting on a row with 219 seats.

e) Write a for-loop which searches through the first 99 rows and attempts to find the row on which Sanna is sitting. You may have use for the following lines of code:

```
for row in range(1, 100):
if seats(row) == 219:
```

9.5 The School Road

145

Code

```
1 def distance_sara(time):  # Distance to the school for Sara
2   return 400 - 1.4*time
3
4 print(distance_sara(100))  # Distance after 100 seconds
```

Sara walks to school every day. She has created the program above to calculate the distance remaining to the school, when she knows how much time has passed since she left home. The time is given in seconds.

a) Write the code above, and run the program. Add the line `print(distance_sara(60))`. What the output of this mean?

Figure 9.4: *Two girls on their way to school.*

b) Guess the result if you add the code `print(distance_sara(0))`. How long is Sara's school road?

c) Guess the result of the following lines of code. What do these answers tell you?

```
print(distance_sara(9) - distance_sara(10))
print(distance_sara(99) - distance_sara(100))
```

d) Write a for-loop to produce a simple table which shows the distance to the school, every tenth second. See table 9.2. Use the table to approximate how long time Sara spends walking to her school.

Time (s)	**Distance to school (m)**
0	400
10	386
20	372
...	...

Tabell 9.2: *Time vs distance to school.*

Pia rides her bike to school with the speed 4,2 m/s. She rides along the same school road as Sara, and starts riding at the same time Sara starts walking. The distance from Pia's house to the school is 600 m.

e) Write a function which calculates the distance to school for Pia. Use the code below as a starting point, and correct it such that the output of `print(avstand_pia(100))` gives 180. *Tip:* Look at the function for Sara's distance to the school.

```
def distance_pia(tid):
  return 4.2*tid + 600  # TODO: correct the calculation

print(distance_pia(100))
```

f) Decide Pia's distance to school after 50 seconds. What is the distance between Pia and Sara at this point?

g) Finally, write code to decide approximately when Pia and Sara meet.

9.6 Deciding the Minimum

146

Code

```
def f(x):
  return x**2 - 2*x + 3

# Print some points
print('(x, y)')
for i in range(-3, 5):
  print(i)
```

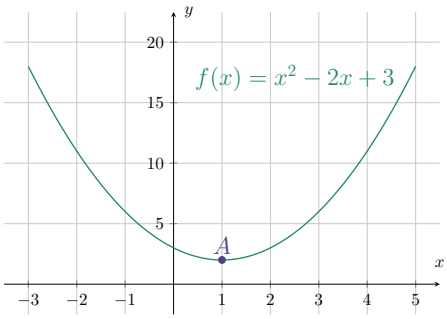


Figure 9.5: *A parabola.*

Figure 9.5 shows the graph of the function $f(x) = x^2 - 2x + 3$.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Add the code `y = f(i)` inside the for-loop. Then change the command `print(i)` to `print(i, y)`, such that both x- and y-values are printed.

d) To print the values in the format of a point, change the `print`-command inside the for-loop to `print(f'({i}, {y})')`.

Recipe

```
Set minimum = f(-4)
Repeat for i = -3, -2, -1, ..., 4:
  Set y = f(i)
  If y < minimum:
    Set minimum = y
Print the value of minimum
```

The recipe above shows an algorithm for finding the minimum[1] of the function.

e) Use the recipe above as a starting point, and finish writing the program such that it finds the minimum.

f) Add the line `x_min = -4` before the for-loop. Use this variable to print the x-coordinate of the minimum, together with the minimum value.

[1] The minimum here means the lowest y-value on the graph of the function $f(x)$.

9.7 A Piece of Modern Art

146

Code

```
1 def value(years):
2   return 500*0.85**years
3
4 buying_price = verdi(0)
5 print(buying_price)
```

$$V(x) = 5000 \cdot 0,85^x$$

Maya buys a modern painting for 500 $. Experts say that the value of the painting will decrease by 15 % every year. The value follows the function $V(x)$, where $V(x)$ is the value in $, x years after Maya bought the painting.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

The command `print(value(1))` gives the value after 1 year, `print(value(2))` gives the value after 2 years and so on.

c) Add code to decide the value of the painting after 5 years.

```
x = 0  # x here means the number of years
while verdien(x) >= 100:
  # TODO: Increment x by 1
# TODO: Print x
```

Maya wishes to keep her painting as long as the value is at least 100 $. Once the value drops below that, she will sell it.

d) Write the code above, and complete it such that the program decides how long Maya will keep her painting.

Figure 9.6: *Modern art. A painting.*

9.8 Summary of Chapter 9

Python	English
`def`	Define a function
`return`	Exits the function
`return 2*5`	Exits the function and returns 10

Tabell 9.3: *Dictionary of chapter 9.*

We have seen how we define typical mathematical functions in Python. Here are a few examples:

```
def f(x):
  return 3*x - 2
```

$$f(x) = 3x - 2$$

```
def g(x):
  return -2*x**2 + 5
```

$$g(x) = -2x^2 + 5$$

```
def h(x):
  return 500/x
```

$$h(x) = \frac{500}{x}$$

```
def s(x):
  return 4000*1.023**x
```

$$s(x) = 4000 \cdot 1,023^x$$

To use the function $s(x)$ above, we can write for example `print(s(5))`. The result is the y-coordinate of the graph of $s(x)$ when $x = 5$.

Note

- In programming we often use descriptive names on functions instead of $f(x)$, $g(x)$ and so on.
- For example, if a function calculates a price for a certain amount of apples, a descriptive name of the function would be `def calculate_price(amount_of_apples):`.

9.9 Exercises

9.9.1 Syntax Error in the Function

147

Code

```
df f(x)=
  3x + 1

print(f(2))
```

The code above is an attempt to define the function

$$f(x) = 3x + 1$$

Correct all the (syntax) errors in the code, and run the program. If the output gives you 7, you did it correctly.

9.9.2 Split the Cost Evenly

147

Figure 9.7: *A wooden cabin.*

Some friends want to rent a cabin. They have decided to split the price evenly. The price is 900 $. Create a program which prints the price per friend, when the number of friends varies from 1 to 20. See table 9.4.

Number of friends	Price per friend
1	900
2	450
3	300
...	...

Tabell 9.4: *Price per friend.*

9.9.3 Cannon

147

Figure 9.8: *Two cannons.*

A cannon fires a shot. The height of the cannon ball is given by the function

$$h(x) = -4,9x^2 + 40x + 1,2$$

where h is the height above the ground in meters, and x is the number of seconds passed after the cannon fired the shot.
Create a program which decides approximately when the cannon ball hits the ground.

9.9.4 The CCP-Virus Spreads

A model[2] for the spread of the CCP-virus in a certain village, is given by:

$$c(x) = \frac{9500}{1 + 2.718^{-0.1x+2.5}}$$

where c is the total cases of CCP-virus infections, x days after the testing began. To determine the number of new infections the first day, calcculate $c(1) - c(0)$. The number of new infections on the fifth day is $c(5) - c(4)$ and so on. Create a program which prints the day number and the number of new infections every day, for the first 120 days. Examine the numbers closely, and explain how the virus spread.
Tip: This is how you define the function in Python:

```
def c(x):
  return 9500 / (1 + 2.718**(-0.1*x + 2.5))
```

[2]Rimmer, Logistics growth model for COVID-19 [8]

CHAPTER 10

Probability

Learning Goals

- *Program simulations of random events.*
- *Explore random events and calculate the probabilities of events, using programming.*

Figure 10.1: *Good luck.*

10.1 Simulation of 100 Births

We will create a program to simulate 100 births. We assume the probability of getting a boy, is the same as the probability of getting a girl. 148

a) Read the code below, and guess the result.

Code

```
1 from random import choice
2
3 baby = choice(['boy', 'girl'])
4 print(baby)
```

b) Write the code on the webpage `replit.com/new/python3`, and run the program.

```
from random import choice

for i in range(10):
  baby = choice(['boy', 'girl'])
  print(baby)
```

We can repeat a trial by using a for-loop.

c) Change the code as shown above. Run the program several times and notice how the output changes.

d) Change `print(baby)` til `print(baby, end=' ')`. Run the program..

e) Finally, change the for-loop such that you simulate 100 births.

10.2 Counting the Girls

148

Code

```
from random import choice

for i in range(10):  # Simulate 10 births
  baby = choice(['boy', 'girl'])
  if baby == 'girl':  # If the baby is a girl
    print(baby)
```

a) Read the code above, and guess the result.

b) Write the code, then run the program five times.

Figure 10.2: *A boy and a girl.*

```
from random import choice

girls = 0
for i in range(10):
  baby = choice(['gutt', 'girl'])
  if baby == 'girl':
    girls += 1

print(f'Girls: {girls}')
```

c) Add the changes in the code as shown above. Run the program a few times.

d) Add the lines of code below at the correct place in the code, such that you also count and print the number of boys.

`boys += 1` `boys = 0`

`else:` `print(f'boys: {boys}')`

e) Change `range(10)` to simulate 10000 births. Let boys and girls start at 1, not 0 (to avoid division by 0). Then calculate and print the ratio $^{\text{girls}}/_{\text{boys}}$ inside the for-loop. What happens with this ratio when we keep increasing the number of births?

10.3 Two Dice

148

Code

```
from random import randint

die1 = randint(1, 6)
print(die1)
```

Figure 10.3: *A die.*

a) Read the code above, and guess the result.

b) Write the code on the webpage `replit.com/new/python3`. Run the program three times.

c) Add the following code, and guess the result.

```
if die1 == 6:
  print('Hurray, you got six!')
```

Run the program until you see the output `Hurray, you got six!`.

d) Add the following code, and guess the result.

```
else:
  print(f'Too bad, you only got {die1}.')
```

e) Add the commands `die2 = randint(1, 6)` and `print(die1, die2)` at the bottom of your code. Delete the command `print(die1)`. Run the program.

f) Change the if-command to `if kast1 == kast2:` to examine whether the dice show an equal number of eyes. Print the output `Two equals!` whenever this happens.

g) Change the output in the else-part, such that if you get two unequal, an output such as `3 and 4, unequal` is printed.

10.4 Many Dice

149

Code

```
from random import randint

throws = 0
while throws < 100:
  die = randint(1, 6)
  throws += 1
  print(f'Number of throws: {throws}')
```

To calculate the probability of an event, we must examine how often the event occurs. Firstly, we need to be able to count the number of die-throws.

a) Read the code above, and guess the result.

b) Write the code, and run the program.

c) Change such that you throw the die 200 times.

d) Add the line `sixes = 0` above the while-loop.

e) Add code such that `sixes` increments by 1 each time the die shows 6. Print the number of times you got 6. These pieces of code may be useful:

```
if die == 6:          sixes += 1

print(f'Number of sixes: {sixes}')
```

To approximately calculate the probability of getting a six, we can examine the ratio

$$\frac{\text{sixes}}{\text{throws}}$$

when we throw the die many times.

f) Add the line `print(sixes / throws)` inside the while-loop. Run the program.

g) Examine what happens with this ratio if the number of throws is 10, 100 and 2000.

h) By calculating $1/6$ on a calculator, we get 0.16666666666666666. How does this correspond with the output you got in g)?

10.5 Balls in a Box – with Replacement

149

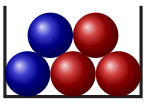


Figure 10.4: *Red and blue balls in a box.*

We put 2 blue and 3 red balls in a box. We are going to pick two balls at random, with replacement. This means that each time we pick a ball, we put it back into the box before we pick again. Let us now use programming to determine the probability of the event "at least one red ball".[1]

[1] Since we are picking two balls, the event "at least one red ball" is equivalent to "one red ball or two red balls".

a) Write the code below, and run the program a few times.

Code

```
from random import choice

balls = 3*['r'] + 2*['b']
print(balls)
ball1 = choice(balls)  # Select one ball at random
print(ball1)
```

b) Pick another ball by adding the line `ball2 = choice(balls)`.

c) Replace the line `print(ball1)` with `print(ball1, ball2)`.

d) Write a for-loop to repeat this trial 10 times.

e) Create a headling by adding the command `print('Ball1 \t Ball2')` before the loop. Change the `print`-command inside the for-loop to `print(f'{ball1} \t \t {ball2}')` to create a nicer looking table:

```
Ball1    Ball2
r        r
r        b
```

f) Use the line `if ball1 == 'r' or ball2 == 'r':` to determine if the event "at least one red" occured.[2] Move `print(f'{ball1} \t \t {ball2}')` inside the if-statement.

g) Change the loop to 1000 repetitions, then count the number of times the event occurs. What is the probability "at least one red ball"?

10.6 Balls in a Box – without Replacement

150

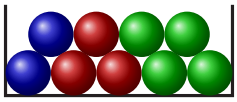


Figure 10.5: *Red, blue and green balls in a box.*

We put 2 blue, 3 red and 4 green balls. Let us pick two balls at random, without replacement. We will explore the event "second ball is not green".

[2]Note that the events "at least one red ball" and "zero blue balls" are equivalent.

a) Write the code below, and run the program a few times..

Code

```
from random import choice

balls = 2*['b'] + 3*['r'] + 4*['g']
print(balls)  # The balls before the 1. pick
ball1 = choice(balls)
balls.remove(ball1)  # Remove the ball from the box
print(balls)  # The balls after the 1. pick
```

b) Add the lines `ball2 = choice(balls)` and `balls.remove(ball2)` at the end of the program, to pick another ball.

c) To see what balls are remaining in the box, add the command `print(balls)`.

d) Add the line `if ball2 == 'g':` to determine if the second ball was green. Print `Second ball was green.`, if the event occured.

The code `not ball2 == 'g'` means "second ball is not green".

e) Change the if-statement to `if not ball2 == 'g':`, and change the corresponding output to `Second ball was not green.`

f) Delete all former `print`-commands, but add the command `print(ball1, ball2)` inside the if-statement.

g) Create a loop to repeat this trial 100 times. An outcome where the second ball is not green, is for example `g b`. How many different such outcomes do you see in the output?

10.7 Two Goats and a Car

150

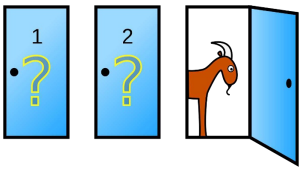


Figure 10.6: *Three dors, two goats and a car.*

A game begins with a participant choosing one of three doors. Behind two of the doors is a goat, and behind the final door is a car.

a) Write the code below, then run the program.

Code

```
from random import choice

doors = ['goat', 'goat', 'car']
door = choice(doors)  # A random door is picked
print(door)
```

b) Use the line `if door == 'car':` to print the output `Congratulations! You won a car!`, if the participant wins the car.

c) Add code such that if the participant does not win the car, the program prints `Unfortunately, you won a goat.`.

After the participant has chosen a door, the game host opens a different door with a goat behind it. The game host then asks the participant if he wants to switch his choice to the other door, or keep his original choice. After the participant makes his final choice, the door he chose is opened.[3]

```
doors.remove('goat')  # The host opens a door with a goat
doors.remove(door)  # Remove the participant's choice
door = choice(doors)  # Choose the remaining door
```

d) Add the code above below the line `print(door)`. Run the program a few times.

e) Write a loop to repeat this trial 20 times. Print an output in every loop, for example `You won a car/goat.`.

f) Create the variable `cars`, and use it to count the number of times the participant wins a car. Print the result after the loop.

g) Change the number of repetitions to 1000. Calculate the ratio below, and print it out.

$$\frac{\text{Number of cars}}{1000}$$

What fraction does this ratio approach?

Finally, what do you think is the best strategy for winning the car? Keep, switch or does it not matter what the participant does?

[3] Based on the game show "Let's make a deal" with the host Monty Hall.

10.8 Shooting Range – Visual Simulation

Code

```
from turtle import *
from random import random

hitrate = 0.6
dot(20, 'black')

if random() < hitrate:
  print('Hit')
  dot(10, 'white')
else:
  print('miss')
```

Figure 10.7: *Shooting range.*

On a shooting range they fire a serises of five shots on five targets.We will create a program to simulate this. In the code above the hit rate is set to 0.6, that is the probability of hitting the target, is 60 %.

a) Read the code above, and guess the result. The command `random()` produces a random number between 0 and 1.

b) Write the code on the webpage `replit.com/new/python_turtle`, and run the program a few times.

c) Add code such that if the shot is a miss, the program shall draw a red circle, instead of a white one.

d) Add code to draw another target. Then write code to simulate shooting on the two targets.
Tip: Use `forward(40)` before you draw target number 2.

e) Finish the program such to simulate the shooting at five targets. We presume the shooter spends two seconds to aim, before he shoots. The result of the program should look like figure 10.7. You may have use for these lines of code:

```
from time import sleep
for i in range(5):
sleep(2)
forward(40)
```

10.9 Summary of Chapter 10

Python	English
`randint(1, 6)`	Random integer between 1 and 6
`random()`	Random number between 0 and 1
`choice(['a', 'b', 'c'])`	Returns one of the elements a, b or c with equal probability
`genders = ['boy', 'girl']`	A list with two elements
`balls = 3*['red'] + 2*['blue']`	List with 3 'red' and 2 'blue'
`choice(balls)`	Choose a random element from `balls`
`balls.remove('red')`	Remove one 'red' from `balls`
`and`	And
`or`	Or
`not`	Not: the opposite

Tabell 10.1: *Dictionary of chapter 10.*

We have seen how to use if-statements to determine if an event occurs. Here are a few examples. We presume the variables used, are already defined.

```
if die1 == 5:
  print('5 dots')
else:
  print('Not 5 dots')
```

```
if die1 == 6 or die2 == 6:
  print('At least one 6')
else:
  print('No sixes')
```

```
if die1 == die2:
  print('Two equal')
```

```
if die1 == 5 and die2 == 5:
  print('Two fives')
```

Note

Write `from random import randint` at the top of the code, to use the command `randint`. Do similarly for `random` and `choice`. You can also import all the commands from the random-library by writing `from random import *`.

10.10 Exercises

10.10.1 Sum of Two Dice

151

```
from random import randint
thesum = die1 + die2
die2 = randint(1, 6)
print(f'The sum is {thesum}')
die1 = randint(1, 6)
```

Write a program to simulate the throwing of two dice. The program should add the dots (eyes) together, and print the sum. Place the lines of code above in the correct order to accomplish this.

10.10.2 Rock Paper Scissors

152

Code

```
1 from random import châise
2 the_choices : [rock, 'paper'. 'scissors']
3 my_choice = choice(theChoices)
4 print(f'You chose {My_choice}')
```

Write a program which picks either rock, paper or scissors at random.[4] Use the code above as a starting points, and correct the errors. The program should print for example `You picked rock`.

10.10.3 Twins

152

Recipe

```
Repeat 9999 times:
  Get pregnant
  If twins:
    Stop getting pregnant

Print the number of children the mother has given birth to
```

Approximately 1 % of births result in twins. A family wishes to have twins, and have decided to continue having new babies until it finally happens. Write a program which follows the recipe above, to simulate this plan. We presume the mother does not give birth to triplings, quadruplings and so on.

[4] We presume uniform probability here, i.e. equal probability for every outcome.

10.10.4 Penalty Kick

152

Figure 10.8: *Football.*

Joe is going to kick 100 penalty kicks. He claims he scores with a 90 % probability. Write a program to simulate the 100 penalty kicks. A possible output:

```
hit
hit
miss
...
```

10.10.5 Smarties

153

Figure 10.9: *Smarties.*

In a bowl lies 5 red and 3 green Smarties. Write a program which simulates that you pick two Smarties from this bowl at random, without replacement. If you get two of equal color, print out `Two equal!`, else print out `One of each`. Repeat the same trial 1000 times, then decide an approximate value for the probability of getting two Smarties of equal color.

10.10.6 Multiple Choice Test

A teacher has created a multiple choice test with 20 questions. Each question has four alternative answers, of which only one is correct. The student will pass the test if he answers correctly on at least 5 questions. Write a program which simulates a student taking this test, but he answers all the questions by guessing wildly. How frequent will such a student, knowing nothing of the subject, pass the test anyways?

APPENDIX A

Løsninger

A.1 Chapter 1 – The First Instructions

1.2 The First Small Steps

```
from turtle import *
shape('turtle')
speed(1)
left(-30)
forward(50)
left(30)
forward(50)
left(30)
forward(50)
```

1.3 Triangle Recipe

```
from turtle import *
shape('turtle')
forward(75)
left(120)
forward(75)
left(120)
forward(75)
left(60)
forward(75)
left(120)
forward(75)
hideturtle()
```

1.4 Square and Rectangle

```
from turtle import *
shape('turtle')
speed(7)
# Draw the rectangle
```

```
forward(200)
left(90)
forward(100)
left(90)
forward(200)
left(90)
forward(100)
left(90)

# Draw the red line segment
forward(100)
left(90)
color('red')
forward(100)
hideturtle()
```

1.5 Drawing with Coordinates

```
from turtle import *
shape('turtle')
# Draw the rhombus
penup()
goto(100, 0)
pendown()
goto(0, 100)
goto(-100, 0)
goto(0, -100)
goto(100, 0)

# Draw the square
goto(100, 100)
goto(-100, 100)
goto(-100, -100)
goto(100, -100)
goto(100, 0)
hideturtle()
```

1.6 Circles and Fillcolors

```
from turtle import *
shape('turtle')
# Draw the largest eye first
fillcolor('lightgray')
begin_fill()
circle(75)
end_fill()
# Draw the light blue eye
fillcolor('lightblue')
begin_fill()
circle(50)
end_fill()
# Finally, draw the small black eye
fillcolor('black')
begin_fill()
circle(25)
end_fill()
hideturtle()
```

1.8.1 Correct Order

```
from turtle import *
shape('turtle')
forward(200)
left(45)
forward(200)
```

1.8.2 Correct the Mistakes

```
from turtle import *
shape('turtle')
forward(300)
left(-180)
forward(300)
```

1.8.3 Zorro

```
from turtle import *
color('gold')
pensize(8)
forward(100)
left(-140)
forward(150)
left(140)
forward(120)
```

1.8.4 Follow the Recipe

```
from turtle import *
shape('turtle')
forward(150)
left(120)
forward(150)
left(120)
forward(150)
left(120)
right(60)
circle(87)
```

1.8.5 Parallelogram

```
from turtle import *
forward(100)
left(60)
forward(130)
left(120)
forward(100)
left(60)
forward(130)
left(120)
hideturtle()
```

A.2 Chapter 2 – Repetitions and Patterns

2.1 Two repetitions

```
from turtle import *
shape('turtle')

for i in range(2):
  circle(50)
  forward(100)
  circle(25)
```

2.2 Three or More Repetitions

```
from turtle import *
shape('turtle')
speed(6)

# Draw the pizza slices
for i in range(8):
  forward(150)
  backward(150)
  left(45)

# Draw the circle
forward(150)
left(90)
circle(150)
```

2.3 Angles and Regular Polygons

```
from turtle import *
shape('turtle')
speed(9)

for i in range(18):  # 18 edges
  forward(25)  # Side length 25
  left(20)  # Angle 20 degrees
```

2.4 Staircase Up and Staircase Down

```
from turtle import *
shape('turtle')
speed(5)

# Draw stairway going up
for i in range(5):
  left(90)
  forward(25)
  right(90)
  forward(50)

# Walk a little forward
forward(50)

# Draw stairway going down
for i in range(5):
  forward(50)
  right(90)
  forward(25)
  left(90)
```

2.7.1 Draw a Square

```
from turtle import *

for i in range(4):
  forward(100)
  left(90)
```

2.7.2 Draw a Rectangle with a For-loop

```
from turtle import *

for i in range(2):
  forward(200)
  left(90)
  forward(100)
  left(90)
```

2.7.3 A Secret Figure

```
from turtle import *

for i in range(8):
  left(45)
  forward(50)
  right(90)
  forward(50)
```

2.7.4 Cones with Icecream

```
from turtle import *
speed(9)

for i in range(10):
  forward(80)
  circle(26)
  backward(80)
  left(36)
hideturtle()
```

A.3 Chapter 3 – Storing Code in Functions

3.1 My smile

```
from turtle import *

# The recipe of a smile
def smile():
  setheading(-90)  # Set heading downward (south)
  circle(50, 180)  # Draw a semicircle with radius 50

def pout():
  setheading(90)
  circle(50, 180)

smile()
smile()
forward(100)
left(90)
```

```
forward(400)
left(90)
forward(100)
smile()
smile()
```

3.2 Cogwheel and Squares

```
from turtle import *
speed(9)

def square():
  for i in range(4):
    forward(50)
    right(90)

for i in range(8):
  square()
  forward(120)
  left(45)
```

3.3 Varying the Size of Squares

```
from turtle import *
speed(7)

def square(side):
  for i in range(4):
    forward(side)
    left(90)

square(50)
backward(25)
square(100)
backward(25)
square(150)
backward(25)
square(200)
hideturtle()
```

3.4 Tunnel in 3D

```
from turtle import *
speed(0)

def square(side):
  for i in range(4):
    forward(side)
    left(90)

sidelength = 200  # Side length begins at 200
while sidelength > 10:
  square(sidelength)
  change = sidelength * 0.03
  sidelength -= change
  setheading(45)
  forward(change * 0.707)  # The factor is really sqrt(2)/2
  setheading(0)
```

3.5 Triangle-pattern with Midpoints

```
from turtle import *
speed(9)

def triangle(side):
  for i in range(3):
    forward(side)
    left(120)

length = 300
for i in range(7):
  triangle(length)
  length /= 2  # Halve the length
  forward(length)
  left(60)
```

3.7.1 A Function to Draw the Letter T

```
from turtle import *
speed(1)

def draw_T():
  forward(100)
  backward(50)
  right(90)
  forward(120)

# Draw T by calling the function
draw_T()
```

3.7.2 Polygon-function

```
from turtle import *
speed(8)

def polygon(edges):
  angle = 360 / edges
  for i in range(edges):
    forward(40)
    left(angle)

polygon(8)
```

3.7.3 A heart

```
from turtle import *

def heart():
  fillcolor('pink')
  begin_fill()
  circle(100, 90)
  circle(50, 180)
  right(180)
  circle(50, 180)
  circle(100, 90)
  end_fill()

```

```
heart()
hideturtle()
```

3.7.4 Eye Monster

```
from turtle import *
speed(9)

def eyemonster():
  circle(25)
  fillcolor('gray')
  begin_fill()
  circle(15)
  end_fill()
  fillcolor('black')
  begin_fill()
  circle(10)
  end_fill()

eyemonster()
forward(75)
eyemonster()
penup()
right(90)
forward(50)
left(90)
pendown()
eyemonster()
backward(75)
eyemonster()
hideturtle()
```

3.7.5 Chessboard – The First Three Squares

```
from turtle import *
speed(0)

def square(color):
  fillcolor(color)
  begin_fill()
  for i in range(4):
```

```
    forward(50)
    left(90)
  end_fill()

square('brown')
forward(50)
square('white')
forward(50)
square('brown')
hideturtle()
```

A.4 Chapter 4 – More Possibilities

4.1 A Wheel with Triangles

```
from turtle import *
speed(9)

side = 80
# Draw the circle
fillcolor('brown')
begin_fill()
circle(side)
end_fill()
penup()
left(90)
forward(side)
pendown()

# Draw the triangles
fillcolor('gold')
begin_fill()
for i in range(8):
  A = pos()
  forward(side)
  left(45)
  forward(side)
  goto(A)
end_fill()
```

4.2 Random Angles

```
from turtle import *
from random import randint
speed(2)

# Draw angle v
angle_v = randint(20, 130)
print(angle_v)
forward(100)
backward(100)
left(angle_v)
forward(100)

# Draw angle u
backward(100)
angle_u = 180 - angle_v
print(angle_u)
left(angle_u)
forward(100)
```

4.3 The Circumscribed Circle of a Triangle

```
from turtle import *
from random import randint

# Draw the circumscribed circle
shape('circle')
u = randint(0, 180)
print(u)
A = pos()

circle(100, u)
B = pos()
stamp()

v = randint(0, 180)
print(v)
circle(100, v)
C = pos()
stamp()

circle(100, 360 - u - v)
```

```
21
22 # Draw the triangle
23 goto(B)
24 goto(C)
25 goto(A)
```

4.4 Bullseye

```
1  from turtle import *
2
3  radius = 150
4  increase = -25
5  for color in ['gray', 'black', 'blue', 'red', 'yellow']:
6    fillcolor(color)
7    begin_fill()
8    circle(radius)
9    end_fill()
10   penup()
11   sety(ycor() - increase)
12   pendown
13   radius += increase
14 hideturtle()
```

4.5 Distance Alarm

```
1  from time import sleep
2  from turtle import *
3  from random import randint
4  Screen().bgcolor('green')
5  shape('circle')
6  speed(1)
7
8  while True:
9    left(randint(0, 359))
10   forward(30)
11   sleep(randint(0, 3))
12   print(distance(home))
13   if distance(home) > 140:
14     Screen().bgcolor('red')
15     print('Come home!')
16     home()
```

```
17      break
18    elif distance(home) > 80:
19      Screen().bgcolor('yellow')
20      print('Mild warning')
```

4.7.1 Scaling with a Variable

```
1  from turtle import *
2
3  radius = 100
4  dot(radius, 'blue')
5  penup()
6  forward(radius * 3)
7  dot(radius * 2, 'brown')
```

4.7.2 Volume-symbol

```
1  from turtle import *
2
3  width(5)
4  radius = 30
5
6  for i in range(8):
7    penup()
8    circle(radius, 30)
9    pendown()
10   circle(radius, 120)
11   penup()
12   circle(radius, -150)
13   radius += 15
14   sety(ycor() - 15)
```

4.7.3 One Circle and 100 Chords

```
1  from turtle import *
2  from random import randint
3  speed(0)
4
5  circle(100)
```

```
for i in range(100):
  circle(100, randint(0, 180))
  A = pos()
  circle(100, randint(0, 180))
  B = pos()
  goto(A)
  goto(B)
```

4.7.4 A watch

```
from turtle import *
speed(0)
shape('circle')

radius = 100
length = radius/4
for i in range(12):
  circle(radius, 360/12)
  left(90)
  forward(length)
  backward(length)
  right(90)

penup()
left(90)
forward(radius)
```

A.5 Chapter 5 – Mathematics in Python

5.6 A Polite Greeting

```
print('Hello! What is your name?')
name = input()  # Let the user type in his name
sex = input('What is your sex?  (male/female): ')

if sex == 'female':
  print(f'Hello, Ms. {navn}')
elif sex == 'male':
  print(f'Hello, Mr. {navn}')
else: # In case the user misspelled male or female
  print(f'Hello, {navn}')
```

5.7 Calculator Program

```
# Read two numbers from the user as decimal numbers (float)
numberA = float(input('Enter a number: '))
numberB = float(input('Enter another number: '))

# Calculate sum, product and quotient
thesum = numberA + numberB
product = numberA * numberB
quotient = numberA / numberB

# Print the results
print(f'{numberA} + {numberB} = {thesum}')
print(f'{numberA} * {numberB} = {product}')
print(f'{numberA} / {numberB} = {quotient}')
```

5.8 The Multiplication Master

```
from random import *

for i in range(10):
  numberA = randint(0, 10)
  numberB = randint(0, 10)
  answer = numberA * numberB
  print(f'{numberA} * {numberB} = ')
  user_answer = int(input('Svar: '))

  if user_answer == answer:
    print('Correct answer!')
  else:
    print('Wrong answer!')
```

5.10.1 The Average of Two Numbers

```
print('Enter a number:')
number1 = float(input())
print('Enter another number:')
number2 = float(input())

thesum = number1 + number2
average = thesum / 2
```

```
print('The average is:')
print(average)
```

5.10.2 How Old Am I?

```
print('Enter your year of birth (e.g. 1982): ')
year = int(input())  # Convert input to a whole number
age = 2021 - year
print(f'You will {age} years old this year.')
```

5.10.3 Currency Converter

```
print('Amount in NOK: ')
nok = float(input())
rate = 9.83  # The conversion rate between USD and NOK
usd = nok / rate
usd = round(usd, 2)
print(f'Amount in USD: {usd}')
```

5.10.4 Temperature Check

```
print('Enter the temperature:')
temp = float(input())

if temp < 0:
  print('Brr')
else:
  print('Getting warmer!!')
```

5.10.5 Map Scale

```
print('Enter distance on the map in inches: ')
distance_map_inches = float(input())
distance_terrain_inches = distance_map_inches * 20000
distance_terrain_miles = distance_terrain_inches / 63360
speed = 4.3  # Unit: miles per hour
hours = distance_terrain_miles / speed

```

```
minutes = round(hours * 60) # Round off to whole minutes
distance = round(distance_terrain_miles, 2)
print(f'Distance in the terrain: {distance} miles')
print(f'Time spent: {minutes} minutes')
```

A.6 Chapter 6 – Number Patterns

6.1 Basic Number Patterns

```
# Print the number pattern 100, 50, 25, 12.5, ...
number = 100
for i in range(10):
  print(number)
  number /= 2
```

6.2 A Fast-Growing Monkey Population

```
monkeys = 4
years = 0

for i in range(30):  # Optional: use a while-loop
  years += 1  # Increment years by 1
  monkeys *= 2  # Double monkeys
  if monkeys > 5000000:  # Greater than 5 million
    print(f'After {years} years')  # 21
    break  # Cancel the for-loop
```

6.3 Sums of Odd Numbers

```
number = 1
thesum = 0
for i in range(8):
  #print(number)  # To check that the final number is 15
  thesum += number
  number += 2
print(thesum)  # 64
```

6.4 Sum of a Geometric Pattern

```
denom = 2  # The first denominator is 2
thesum = 0
for i in range(60):
  area = 1 / denom
  thesum += area
  print(thesum )  # The sum approaches 1
  denom *= 2
```

6.5 Number Pattern with Circles

```
from turtle import *

speed(0)
radius = 100
while radius > 0:
  circle(radius)
  circle(-radius)
  forward(radius)
  radius -= 10
  forward(radius)
hideturtle()
```

6.6 The First 100 Natural Numbers

```
from turtle import *
speed(9)
penup()
backward(200)
pendown()

area_sum = 0
for number in range(101):
  print(number)
  # The area of a rectangle is number * 1 = number
  area_sum += number
  sety(number)  # Draw the bar
  sety(0)  # Go back to the x-axis
  forward(1)

```

```
print('The area is: ')
print(area_sum)
```

6.7 Rectangle Pattern

```
from turtle import *
speed(8)
fillcolor('orange')

def rectangle(width, height):
  begin_fill()
  for i in range(2):
    forward(width)
    left(90)
    forward(height)
    left(90)
  end_fill()

height = 20
for i in range(8):
  rectangle(10, height)
  penup()
  forward(20)
  pendown()
  height += 30
```

6.9.1 The 4 Times Table

```
for number in range(1, 11):
  product = number * 4
  print(product)
```

6.9.2 Number Pattern with 100 Terms

```
number = 10
# Term number from 1 to 100 (both inclusive)
for term_nr in range(1, 101):
  print(term_nr, number)
  number += 3  # Increment number by 3
```

6.9.3 Dots in Circles

```
s = 1
for term_nr in range(1, 16):
  print(term_nr, s)  # s_15 = 16384
  s *= 2
```

6.9.4 Number Pattern by Recipe

```
number = 100000
halves = 0
while number >= 0.1:
  #print(number)  # Optional
  number /= 2
  halves += 1
#print(number)  # Optional
print(f'Number of halves: {halves}')  # 20
```

6.9.5 Number of Terms in the Sum

```
term_nr = 0
thesum = 0
while thesum <= 200000:
  term_nr += 1
  number = term_nr * 7
  #print(number)  # For testing purposes
  thesum += number
print(term_nr)  # 239
```

6.9.6 Sum of an Infinite Number of Terms

```
thesum = 0
number = 1000
while number > 10 ** (-15): # Stops once number gets tiny
  thesum += number
  number *= 0.9
print(thesum )  # 10000
```

6.9.7 To Run a Marathon with Decreasing Speed

```
distance = 42  # Kilometers
speed = 15  # Kilometers per hour
completed = 0  # Distance completed
hours = 0
while completed < distance:
  completed += 1
  hours += 1 / speed
  print(hours)
  speed -= 0.1

hours = round(hours, 2)
print(f'Stein spends {hours} hours.')  #  3,27h (3h16min)
```

A.7 Chapter 7 – Powers and Square Roots

7.1 Practice Powers of 2

```
from random import randint
print('Test yourself in powers of 2')
base = 2
correct_answers = 0

while correct_answers < 5:
  exponent = randint(-2, 5)
  print(f'{base} ^ {exponent} = ', end='')
  user_answer = float(input())
  answer = base ** exponent
  if user_answer == answer:
    print('Correct!')
    correct_answers += 1
  else:
    print('Wrong. ', end='')
    print(f'Correct answer: {answer}')

print('Congratulations! You answered correctly 5 times!')
```

7.2 Explore Powers of 10

```
print('Exploration of powers of 10.')  # A headline
print()  # Create a blank line
base = 10
for exponent in range(-15, 0):
  number = base ** exponent
  decimals = exponent * (-1)
  answer = f'{number:.{decimals}f}'
  answer_text = f'{base} ^ {exponent} \t = \t {answer}'
  print(answer_text)
```

7.3 Guess the Square Root

```
from random import randint
from math import sqrt

number = randint(1, 101)
answer = sqrt(number)
answer = round(answer, 1)
print(f'Guess the square root of {number}: ')

while True:
  guess = float(input())
  #print(f'You guessed {guess}. Correct answer: {answer}.')
  if guess < answer:
    print('You guessed too low.')
  elif guess > answer:
    print('You guessed too high.')
  else:
    print('Correct!')
    break
```

7.4 Rice and Chess

```
thesum = 0
for square_nr in range(64):
  rice = 2 ** square_nr
  thesum += rice
  #print(rice)
print(thesum)
```

7.6.1 Square Numbers

```
for base in range(1, 21):
  number = base ** 2
  print(number)
```

7.6.2 Cubic Numbers

```
term_nr = 1
cubic = term_nr ** 3
while cubic < 1000000:
  print(cubic)
  term_nr += 1
  cubi = term_nr ** 3
print(term_nr)  # 100, since 100*100*100 = 1 million
```

A.8 Chapter 8 – The Coordinate System

8.2 Explore *x*-coordinates with Python Turtle

```
from turtle import *
shape('arrow')

def draw_xaxis():
  circle(10)  # Draw a point where x = 0
  setx(-220)
  setx(220)
  stamp()  # Place the arrow for the x-axis

draw_xaxis()  # Run the lines 5 through 8

for i in range(3):
  print('Enter a x-coordinate (-200 to 200):')
  x_coord = int(input())
  setx(x_coord)
  if x_coord < 0:
    dot(10, 'red')
  else:  # Draw a blue circle if x is positive
    dot(10, 'blue')
```

8.3 Explore *y*-coordinates with Python Turtle

```
from turtle import *
shape('arrow')

def draw_yaxis():
  left(90)
  sety(-120)
  sety(150)
  stamp()

draw_yaxis()
sety(0)
sety(-120)  # Move the arrow to the bottom of the y-axis
setheading(0)  # Set heading rightward

for i in range(14):
  y = ycor()  # Store y-coordinate in the variable y
  if y < 0:
    color('red')
  else:  # Blue color if y is positive
    color('blue')
  forward(y)
  backward(y)
  color('black')  # Black color along the axis
  sety(y + 20)  # Increment y-coordinate by 20

hideturtle()
```

8.4 Parallel Lines

```
from turtle import *
speed(9)

# Draw horizontal lines
penup()
sety(-150)
for i in range(11):
  pendown()
  setx(300)
  setx(0)
  penup()
  sety(ycor() + 30)
```

```
13
14 sety(150)
15 # Draw vertical lines
16 for i in range(11):
17   pendown()
18   sety(150)
19   sety(-150)
20   penup()
21   setx(xcor() + 30)
```

8.6.1 The Letter T – Again

```
1 from turtle import *
2
3 sety(100)
4 setx(45)
5 setx(-45)
6 hideturtle()
```

8.6.2 An Unknown Geometric Figure

```
1 from turtle import *
2
3 print('Enter the x-coordinate: ')
4 x = float(input())
5 print('Enter the y-coordinate: ')
6 y = float(input())
7
8 # Draw a parallelogram
9 goto(x, y)
10 sety(-y)
11 goto(0, -2 * y)
12 goto(0, 0)
```

8.6.3 The Distance to a Point

```
1 from math import sqrt
2 from turtle import *
3 speed(2)
```

```
4
5  print('Enter the x-coordinate: ')
6  x = float(input())
7  print('Enter the y-coordinate: ')
8  y = float(input())
9  goto(x, y)
10
11 distance = sqrt(x**2 + y**2)
12 print('Distance to the point: ')
13 print(distance)
```

A.9 Chapter 9 – Functions

9.1 Kaare on the Café – without a Function

```
1  price_per_cup = 4
2  print('How many cups of coffee do you want?')
3
4  cups = int(input())  # Convert input to whole number
5  price = price_per_cup * cups
6  print(f'Price: {price} dollars')
```

9.2 Kaare on the Café – with a Function

```
1  def calculate_price(cups):
2    return cups * price_per_cup
3
4  price_per_cup = 4
5  print('Cups\t Price in dollars')
6  for my_cups in range(1, 81):
7    price = calculate_price(my_cups)
8    #print(f'{my_cups} cups cost {price} dollars.')
9    print(f'{my_cups} \t\t {price} ')  # 320 dollars for 80 cups
```

9.3 Price per Workout

```
def y(x):
  return 50 / x

print('Number of workouts \t Price per workout')
workouts = 1
for i in range(5):
  price = y(workouts)
  print(f'{workouts}\t\t\t\t {price}')
  workouts *= 2
# When you double the number of workouts, the price
# per workout is halved.
```

9.4 Seats in a Cinema

```
def seats(row_nr):
  return 4*row_nr - 1

# Number of seats on row 1
print(seats(1))
# The number of extra seats on row 11 compared to row 10
print(seats(11) - seats(10))

# Search through row 1 to 99
for row in range(1, 100):
  if seats(row) == 219:
    print('Sanna is sitting on row number:')
    print(row)  # 55
```

9.5 The School Road

```
def distance_sara(time):  # time in seconds
  return 400 - 1.4*time

def distance_pia(time):
  return 600 - 4.2*time

print('Time\t Distance between Sara and Pia')
for sec in range(80):  # Where sec means seconds
  distance_between = distance_pia(sec) - distance_sara(sec)
```

```
  distance_between = round(distance_between, 1)
  print(f'{sec}\t {distance_between}')
# Read the answer from the output: between 71 and 72 seconds
```

9.6 Deciding the Minimum

```
def f(x):
  return x**2 - 2*x + 3

# Two variables for the coordinates
minimum = f(-4)  # y-coordinate
x_min = -4  # x-coordinate

for i in range(-3, 5):
  y = f(i)
  if y < minimum:
    minimum = y
    x_min = i

print('The minimum is:')
print(f'({x_min}, {minimum})')  # (1, 2)
```

9.7 A Piece of Modern Art

```
def value(years):
  return 500*0.85**years

buying_price = value(0)
print(buying_price)
print(value(5))  # The value after 5 years

x = 0  # x represents the number of years passed
while value(x) >= 100:
  x += 1  # Increment the number of years by 1
  print(value(x))  # Optional line of code

print(x)  # 10
```

9.9.1 Syntax Error in the Function

```
def f(x):
  return 3*x + 1

print(f(2))
```

9.9.2 Split the Cost Evenly

```
def calculate_price(number_of_friends):
  return 900 / number_of_friends

print('Number of friends \t Price per friend')

for friends in range(1, 21):
  price = calculate_price(friends)
  print(f'{friends}\t\t\t\t\t {price}')
  #print(f'{friends}\t\t\t\t\t {price:.2f}') # Alternatively
```

9.9.3 Cannon

```
# h is the height above the ground in meters
# x is the number of seconds
def h(x):
  return -4.9*x**2 + 40*x + 1.2

sec = 0  # Time in seconds
height = h(0)  # Height in the beginning

while height > 0:  # As long as the ball is airborne
  sec += 0.1  # Increment the time by 0.1 seconds
  height = h(sec)  # Calculate the height again
  #print(f'{sec}\t\t {height}')  # Optional

print(f'The ball hits the ground after {sec:.2f} seconds.')
# Answer: 8.20 seconds
# tid:.2f rounds off to two decimal places
```

A.10 Chapter 10 – Probability

10.1 Simulation of 100 Births

```
from random import choice

for i in range(100):
  baby = choice(['boy', 'girl'])
  print(baby, end=' ')
```

10.2 Counting the Girls

```
from random import choice

girls = 1
boys = 1

for i in range(10000):
  baby = choice(['boy', 'girl'])
  if baby == 'girl':
    girls += 1
  else:
    boys += 1
  print(girls/boys) # The ratio approaches 1
```

10.3 Two Dice

```
from random import randint

die1 = randint(1, 6)
die2 = randint(1, 6)
print(die1, die2)

if die1 == die2:
  print('Two equal!')
else:
  print(f'{die1} and {die2}, two unequal.')
```

10.4 Many Dice

```
from random import randint

throws = 0
sixes = 0

while throws < 5000:
  die = randint(1, 6)
  throws += 1
  if die == 6:
    sixes += 1
  print(sixes / throws)

print(f'Number of throws: {throws}')
print(f'Number of sixes: {sixes}')
```

10.5 Balls in a Box – with Replacement

```
from random import choice

balls = 3*['r'] + 2*['b']
print(balls)
counter = 0  # The number of times we get at least one red
trials = 1000
print('Ball1 \t Ball2')

for i in range(trials):
  ball1 = choice(balls)  # Pick a ball at random
  ball2 = choice(balls)
  print(f'{ball1} \t \t {ball2}')
  if ball1 == 'r' or ball2 == 'r':
    counter += 1

print(f'At least one red: {counter} times')
relative_frequency = counter / trials
print(f'Calculated probability: {relative_frequency}')  # 0.83
```

10.6 Balls in a Box – without Replacement

```
from random import choice

for i in range(100):
  balls = 2*['b'] + 3*['r'] + 4*['g']
  ball1 = choice(balls)
  balls.remove(ball1)
  ball2 = choice(balls)
  balls.remove(ball2)
  if not ball2 == 'g':
    print(ball1, ball2)
# 6 different outcomes: rr, rb, br, gr, gb and bb
```

10.7 Two Goats and a Car

```
from random import choice

cars = 0
trials = 1000

for i in range(trials):
  doors = ['goat', 'goat', 'car']
  door = choice(doors)  # A random door is picked
  #print(door)
  doors.remove('goat')  # The host opens a goat-door
  doors.remove(door)    # Remove the original choice
  door = choice(doors)  # Choose the remaining door

  if door == 'car':
      cars += 1
  #print(f'You won a {door}')  # Optional

print(f'Cars won: {cars}')
p = cars / trials
print(f'Win rate: {p}')  # Close to the fraction 2/3
```

10.8 Shooting Range – Visual Simulation

```
from turtle import *
from random import random
from time import sleep

hitrate = 0.6
penup()
hideturtle()

# Draw the targets
for i in range(5):
  dot(20, 'black')
  forward(40)

home()
# Simulate the shooting
for i in range(5):
  sleep(2)
  if random() < hitrate:
    print('Hit')
    dot(10, 'white')
  else:
    print('Miss')
    dot(10, 'red')
  forward(40)
```

10.10.1 Sum of Two Dice

```
from random import randint

die1 = randint(1, 6)
die2 = randint(1, 6)
thesum = die1 + die2
print(f'The sum is {thesum}')
```

10.10.2 Rock Paper Scissors

```
from random import choice

choices = ['stein', 'saks', 'papir']
my_choice = choice(choices)
print(f'You chose {my_choice}')
```

10.10.3 Twins

```
from random import randint

children = 0

for i in range(9999):
  pregnant = randint(1, 100)
  children += 1
  if pregnant == 100:  # Twins, 1 of 100
    children += 1  # Count the extra child
    break

print(f'Number of children: {children}')
```

10.10.4 Penalty Kick

```
from random import random

hitrate = 0.9

for i in range(100):
  if random() < hitrate:
    print('hit')
  else:
    print('miss')
```

10.10.5 Smarties

```python
from random import choice

equals = 0
trials = 1000

for i in range(trials):
  smarties = 5*['r'] + 3*['g']
  smartie1 = choice(smarties)
  smarties.remove(smartie1)
  smartie2 = choice(smarties)
  smarties.remove(smartie2)
  if smartie1 == smartie2:
    print('Two equal!')
    equals += 1
  else:
    print('One of each.')

p = equals / trials
print(f'Probability of two equal: {p}')  # 0.46
```

APPENDIX B

Plotting Graphs in Python with Spyder

B.1 Installation of Anaconda/Spyder

We will use the popular software Spyder to plot some graphs in Python. Spyder is a part of the Anaconda-installation.

a) Open the webpage `www.anaconda.com`.

b) Navigate to the download link: Products » Individual Edition » Download.

c) Choose the download link dependant upon your operating system (Windows, MacOS or Linux).

d) Install Anaconda.

When the installation of Anaconda is complete, you are ready to use Spyder.

B.2 Your First Python-program in Spyder

We will create a Python-program in Spyder to plot a graph through four points. To plot graphs, we will use a library called *matplotlib*.

a) Open the program Spyder.

b) Create a new file from the menu like this: File » New file...

c) If you want to save your file, go to File » Save as... Note that the name of Python-files must end with `.py` to work.

When you create a new file in Spyder, the start of the file looks like this:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Mar 14 13:37:42 2021
4
5 @author: thomassowell
6 """
```

All this can be safely be deleted, before you start writing your program.

d) Begin by writing the code below.

Code

```
1 import matplotlib.pyplot as plt
2
3 x_values = [0, 2]
4 y_values = [1, 3]
5 plt.plot(x_values, y_values)
```

This program plots a line segment from the point $(0, 1)$ to the point $(2, 3)$.

e) Save your program: File » Save.

f) Run the program by pressing F5 , or use the menu: Run » Run. You should now see the line segment.

g) Add the point $(3, -2)$ by changing the `x_values` and `y_values`:

```
x_values = [0, 2, 3]
y_values = [1, 3, -2]
```

Run the program.

h) Similarly, add the point $(5, 0)$. The result should look like figure B.1.

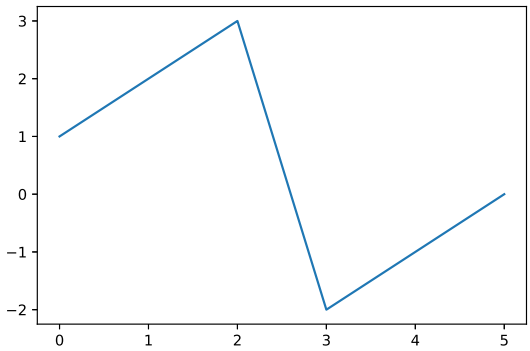


Figure B.1: *My first graph in Python.*

Code of Figure B.1

```
import matplotlib.pyplot as plt

x_values = [0, 2, 3, 5]
y_values = [1, 3, -2, 0]

plt.plot(x_values, y_values)
```

B.3 Draw the Graph of a Function

We will now see how we can draw the graph of a function. We choose the function $h(x) = -4,9x^2 + 40x + 1,2$ from exercise 9.9.3 on page 103.

a) Write the code below.

```
import matplotlib.pyplot as plt
import numpy as np

x_values = np.arange(1, 6)
print(x_values)
```

b) Save the file, then run the program.

From the *numpy*-library we can use the command `arange`. This command can create many x-values.

c) Change `np.arange` to `np.arange(-4, 5)`. Guess the result, then run the program.

d) Change `np.arange` to `np.arange(-4, 5, 2)`.

The line `x_values = np.arange(-4, 5, 2)` gives us the x-values $-4, -2, 0, 2$ and 4. The distance between the x-values is 2.

e) Calculate the corresponding y-values and plot the graph of the function $h(x) = -4,9x^2 + 40x + 1,2$ by adding the lines of code below.

```
y_values = -4.9*x_values**2 + 40*x_values + 1.2
plt.plot(x_values, y_values)
```

Maybe you notice that the curve is uneven. To draw a more even curve, the distance between the x-values needs to be smaller.

f) Change `np.arange` to `np.arange(-4, 5, 0.1)`. Guess the result, before you run the program.

The cannon ball was in the air for 8.2 seconds before it hit the ground. Hence, the x-values should start at 0 and end at 8.2.

g) Change `np.arange` such that the graph is plotted only for the x-values from 0 til 8.2. You should now get figure B.2.

The variable names `x_values` and `y_values` are useful in the beginning, to understand that they contain not only one value, but several values. When you feel comfortable with this, you can simplify the names to `x` `y`.

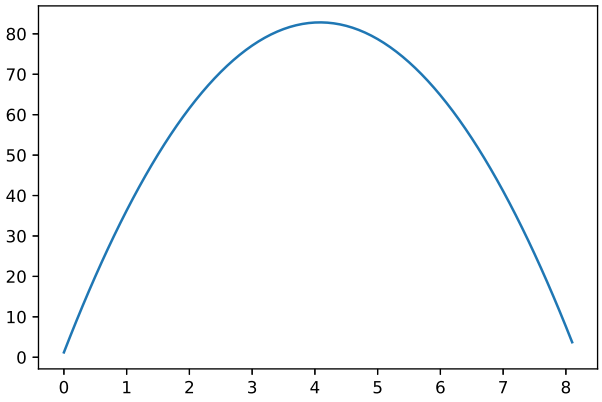


Figure B.2: *The graph of a second degree polynomial.*

Code of FigureB.2

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.arange(0, 8.2, 0.1)
5 y = -4.9*x**2 + 40*x + 1.2
6 plt.plot(x, y)
```

B.4 Axis Grid and Title

Code

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 8.2, 0.1)
y = -4.9*x**2 + 40*x + 1.2
plt.plot(x, y)
```

We continue with the graph of the function $h(x) = -4,9x^2 + 40x + 1,2$, where h is the height of the ball in meters above the ground, and x is the number of seconds. Let's add axis, a grid and a title.

a) Add the line `plt.xlabel('Time (s)')` to create a label for the x-axis.

b) Add the line `plt.ylabel('Height (m)')` to create a label for the y-axis.

c) Create a grid by adding the line `plt.grid()`.

d) You can also add a descriptive title to your graph. Do this by adding the line `plt.title('The Trajectory of a Cannon Ball.')`. You should now see figure B.3.

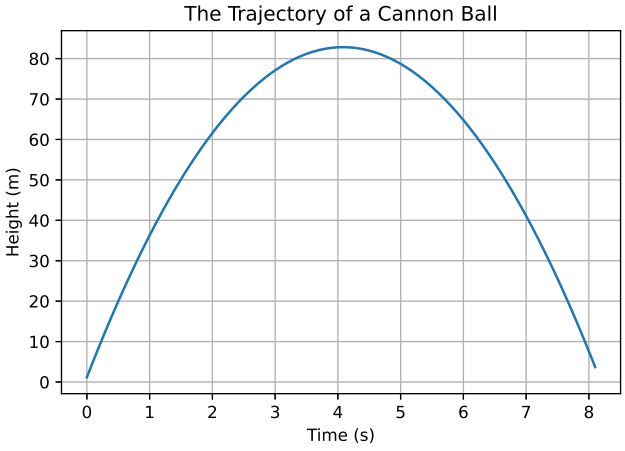


Figure B.3: *A plot with axis, a grid and a title.*

B.5 Complete Example with Two Graphs

Code

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 200, 0.1)

# Calculate the y-values to each function
income = 120*x - 0.3*x**2
expenses = 0.15*x**2 + 40*x + 1700

# Plot the graphs of the functions
plt.plot(x, income, label = 'Income')
plt.plot(x, expenses, label = r'$K(x)=0.15x^2+40x+1700$')

# Add titles, grid and labels
plt.xlabel('Number of units')
plt.ylabel('Kr')
plt.grid()
plt.title('Income and Expenses')
plt.legend()
```

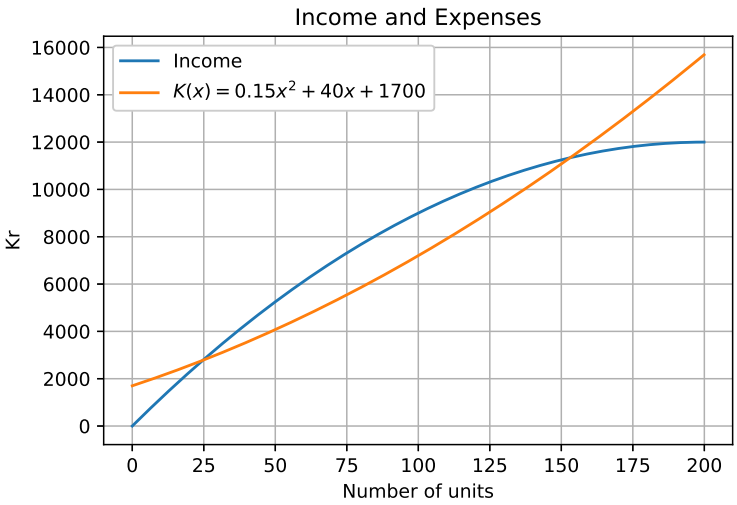


Figure B.4: *Two graphs drawn in Python.*

B.6 Trigonometric Functions

Code

```
import matplotlib.pyplot as plt
import numpy as np

x_degrees = np.arange(0, 360, 0.1)
# Convert degrees to radians
x_radians = np.deg2rad(x_degrees)

# Calculate sine and cosine to all the angles
sin_values = np.sin(x_radians)
cos_values = np.cos(x_radians)

plt.plot(x_degrees, sin_values, label = r'$f(x)=\sin x$')
plt.plot(x_degrees, cos_values, label = r'$g(x)=\cos x$')

plt.xlabel(r'Angle ($\degree$)')
plt.ylabel('Function Value')
plt.grid()
plt.title('Sine and Cosine to the First 360 Degrees')
plt.legend()
```

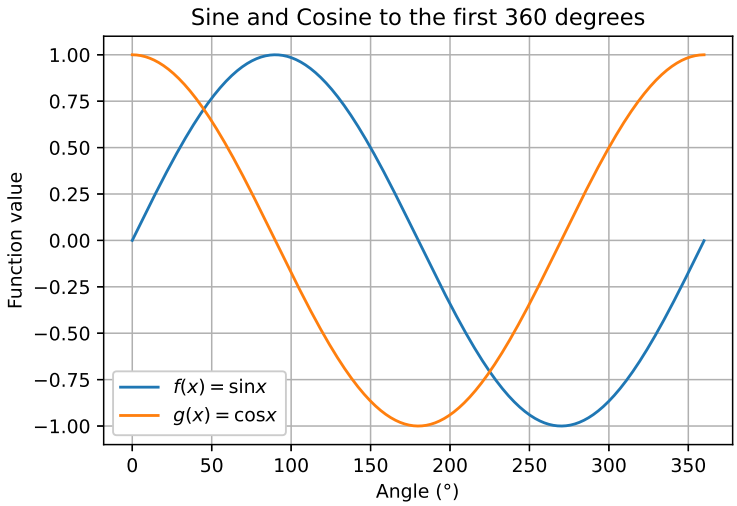


Figure B.5: *The graphs of sine and cosine.*

APPENDIX C

The Learning Philosophy

In this chapter we will present the learning philosophy of this book. First we will tell you shortly about the programming language Python. Then we will describe visual learning and the advantages with this. We present five good tips for learning programming with this book. Finally, we will tell you more in depth about the learning philosophy.

What is Python?

Python is a programming language created by Guido van Rossum in 1991. It is one of the most popular languages, especially among the natural sciences in universities, colleges and schools around the world. Python is well suited for beginners, since the unique simple syntax focuses on readability.

Python Turtle

Python Turtle is a library which contains many functions and commands. Turtle lets us control a pen which can draw on the computer screen. We can draw lines, triangles, circles, railroads and whatever else we feel like drawing. The first four chapters of this book use only Python Turtle to give the reader a visual approach to learning, instead of a text-based approach.

Why Visual Learning?

In many programming courses or books, the first example program will often look like this:

My First Program

```
class HelloWorldApp {
    public static void main(string [] args) {
        System.out.println("Hello, World!");
    }
}
```

The result of the program is that it prints the text `Hello, World!` on the screen. This is how foregår en **text-based learning** of programming works. **Visual learning** by using Python Turtle, produces results of code as visual figures, instead of text. See figure C.1.

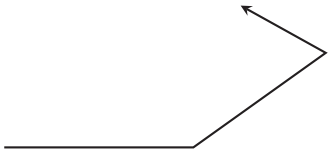


Figure C.1: *A drawing – my first program with Python Turtle.*

Visual learning means that you can write just on line of code, and produce for example a circle. This invites the reader to a whole new world of exploration and experimentation. A Danish study showed that when the focus is on immediate visual feedback, the students motivation and ability to solve problems significantly increases.[1]

How can you learn programming with this book?

Here are five pieces of advice to get you started:

1) **First read the example.** Always try to guess the result of the code, before you run the program.

2) **Write the code provided in the example.** It is very important you write code yourself, to get the feel of it and get the coding fingers running along. Then run the program.

3) **Complete all the steps in the section.** Each section has its own recipe, which contains small steps you need to complete to improve your programming skills.

4) **Try and fail!** Never be afraid to try out other numbers, commands and codes than what is written in the book. We encourage you to experiment with everything you learn, and try other approaches.

5) **Be creative.** Apply what you have learned to new situations. Did you just learn the code to draw a rectangle? Can you apply this knowledge to draw a triangle or a parallelogram?

[1] Reng, [...] Direct Visual Feedback [7]

Our Approach to Learning Programming

This book follows a structured approach to learning programming. In short, the structure is called *PRIMM*[2] and this is what it means:

Predict. Read the code, and predict what the result will be.

Run. Write the code, then run the program. How did the result corresponding to your earlier guess?

Investigate. In this step you will make small changes to the code, such as changing a few values, or changing the order of two lines of code. When you do this step, it is important that you once again repeat the first two steps, that is to predict the result, then run the program again.

Modify. In this step you will make bigger changes to your code. You will expand the program by adding several new lines of code, or you will finish a program to solve a particular problem. You are now on your way to creating your own program.

Make. We consider this step the highest level of achievement. Here you are on your own, and will create your own program all by yourself.

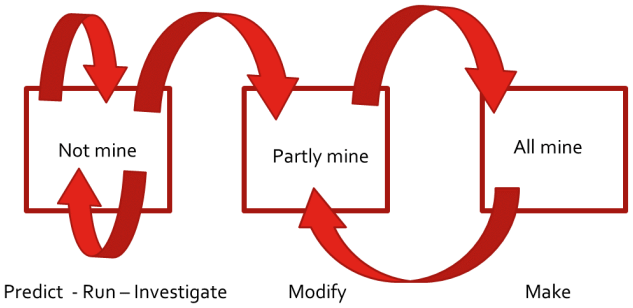


Figure C.2: *PRIMM-diagram.*

The PRIMM-diagram[3] shows how the students work their way through three competency levels in programming. At first they are playing with a program which is not theirs, and they read the code and test the program. In the next phase they will make bigger changes to their code, and the program is now partly theirs. Finally, they create a whole new program which is completely their own product. This is the highest competency level.

[2]Sentance, Primm - a structured approach to teaching programming [11]
[3]Sentance, Exploring pedagogies for teaching programming in school [10]

Algorithmic Thinking and Different Representations

> "Algorithmic thinking is a problem solving method. Algorithmic thinking involves approaching problems on a systematic way, both when we formulate what we wish to solve and when we suggest possible solutions."
>
> *Norwegian Directorate for Education and Training*[4]

This is how the Norwegian Directorate for Education and Training describes the term *algorithmic thinking*. One of the systematic ways to approach problems, is to make use of different representations. We use the following number pattern as an example:

$$3, 5, 7, 9, \ldots$$

In figure C.3 we see typical mathematical representations of this.

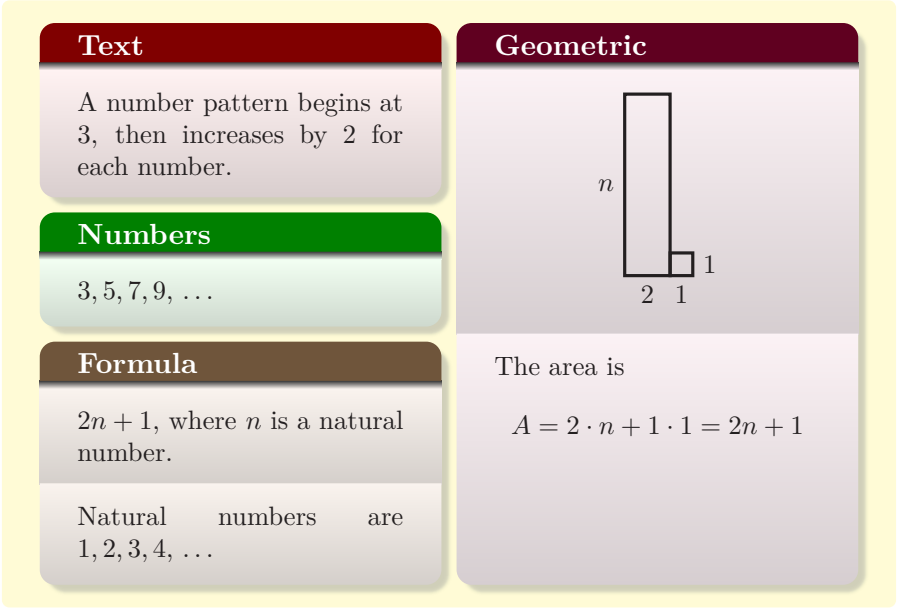


Figure C.3: *Representations in mathematics.*

[4]Education and Training, Algoritmisk tenkning [1]

Programming and algorithmic thinking adds three new representations.

- **Recipe/algorithm**: A recipe, just like an algorithm, involves a set of stepwise instructions. We write the recipe in English.
- **Code**: The code is written in Python, and consists of instructions we give to the computer.
- **Images and figures:** The result of the code can be something **visual**, such as an image or a figure.

These new representations in figure C.4.

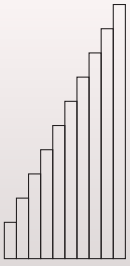
Recipe (algorithm)

```
Let the number a start at 3
Repeat step 3 and 4:
  Print a
  Increment a by 2
```

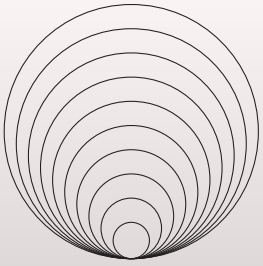
Code

```
a = 3
for i in range(10):
  print(a)
  a = a + 2
```

Figures



(a) Number pattern with rectangles.



(b) Number pattern with circles.

The first rectangle has height 3, the next height 5 and so on. The circles start with radius 3, then the radius increases by 2 for each new circle.

Figure C.4: *New representations in programming.*

Problem Solving with Programming

> "If you can't solve a problem, then there is an easier problem you can solve: find it."
>
> *George Pólya*[5]

This book trains its readers to follow Pólyas advice. Here is an example from chapter 2 where the reader is asked to draw a railroad track:

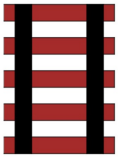


Figure C.5: *A railroad track. Problem solving.*

The problem the student is asked to solve, is to draw the railroad track. Many students will not be able to solve this problem directly. But given the figure C.5, it becomes intuitive for the student to split the problem in smaller and solvable parts. A smaller problem, is for example to draw the bottom rectangle. Next, to complete all the horizontal rectangles. And, finally, color all the rectangles.

Motivation

Finally we would like to share some motivating and inspiring words from an author and professor, who first introduced us to the visual approach to programming:

> "This book tells a story. It's a story of liberation, of taking the first steps towards understanding the foundations of computing, writing your own code, and creating your own media without the bonds of existing software tools. This story is not reserved for computer scientists and engineers. This story is for you."
>
> *Daniel Shiffman*[6]

[5] Pólya, Mathematical Discovery, Volume I [5]
[6] Schiffman, Learning Processing 2nd Edition [9]

List of Images

Description	Page	Source
Road red/blue	Cover	Pixabay/Pexels
Turtle	3	Pixabay/Pexels
Zorro	11	ben.hannis/Flickr
Railroad	13	kalhh/Pixabay
Lady	23	3652586/Pixabay
Cogwheel	25	qimono/Pixabay
Jesus in a window	38	hans-2/Pixabay
Target	39	QuinceCreative/Pixabay
Boy in a raincoat	41	michael podger/Stocksnap
Watch	45	Jaelynn Castillo/Unsplash
Compass	46	OpenClipart-Vectors/Pixabay
Red apples	51	krishnam_moosaddee/Unsplash
Thermometer	58	publicdomainpictures-14/Pixabay
Two monkeys	64	Free-Photos/Pixabay
Fractals	67	realworkhard/Pixabay
Marathon runner	74	kinkate/Pixabay
Rubik's cube	77	OpenClipart-Vectors/Pixabay
Chessboard and rice	82	McGeddon/WikiMedia
Cup of coffee	93	Free-Photos/Pixabay
Walter works out	95	wagjm/Pixabay
Cinema	96	fedotov_vs/Unsplash
Girls walking	97	MaBraS/Pixabay
Modern art	100	steve/Pexels
Wooden cabin	102	OpenClipart-Vectors/Pixabay
Two cannons	103	David Mark/Pixabay
Good luck	105	Alexas_Fotos/Pixabay
Boy and girl	106	kevinbgent/Unsplash
Die	107	glencarrie/Unsplash
Three dors	111	Cepheus/Wikimedia Commons
Football	116	S. Hermann & F. Richter/Pixabay
Nonstop	116	ChickenFalls/Wikimedia Commons
Turtle-symbol	Many places	vecteezy.com

List of Figures

Python-words

Bibliography

[1] N. D. for Education and Training. Algoritmisk tenkning. [22.03.2021]. URL: https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/.

[2] Euclid. Euclids Elements. [Translated by Thomas L. Heath]. Green Lion Press, 2013.

[3] M. Goossens et al. The LaTeX Companion. Reading, Massachusetts: Addison–Wesley, 1993.

[4] H. R. Jacobs. Geometry: Seeing, Doing, Understanding. Master Books, 2017.

[5] G. Pólya. Mathematical Discovery, Volume I. Ishi Press, 2009.

[6] D. Prager. Happiness is a serious problem. ReganBooks, 1998.

[7] L. Reng. [...] Direct Visual Feedback. [22.03.2021]. Aalborg University. URL: https://vbn.aau.dk/files/78681073/Enhancing_Students_Motivation_to_Learn_Programming_by_Using_Direct_Visual_Feedback_Innovations2012.pdf.

[8] D. B. Rimmer. Logistics growth model for COVID-19. [22.03.2021]. URL: https://www.wolframcloud.com/obj/covid-19/Published/Logistic-Growth-Model-for-COVID-19.nb.

[9] D. Schiffman. Learning Processing 2nd Edition. [22.03.2021]. URL: http://learningprocessing.com/.

[10] S. Sentance. Exploring pedagogies for teaching programming in school. [22.03.2021]. URL: https://blogs.kcl.ac.uk/cser/2017/02/20/exploring-pedagogies-for-teaching-programming-in-school/.

[11] S. Sentance. Primm - a structured approach to teaching programming. [22.03.2021]. URL: https://blogs.kcl.ac.uk/cser/2017/09/01/primm-a-structured-approach-to-teaching-programming/.

[12] T. F. Sturm. The tcolorbox package. [22.03.2021]. Institut für Mathematik und Informatik, Universität der Bundeswehr München. URL: https://ctan.uib.no/macros/latex/contrib/tcolorbox/tcolorbox.pdf.

[13] T. Tantau. The Tikz and PGF manual. [22.03.2021]. Institut für Theoretische Informatik Universität zu Lübeck. URL: https://ctan.uib.no/graphics/pgf/base/doc/pgfmanual.pdf.