

Les 1: Programmeren als een stappenplan

Het opstellen van een programma komt neer op het neerschrijven van een gedetailleerd stappenplan. In deze eerste les bekijken we volgende controlestructuren van Python:

- Sequentiële uitvoering van commando's;
- Herhalingslussen:
 - *while-loop*
- Conditionele uitvoering *if-elif-else*;
- Variabelen;

Commando's

In ons eerste voorbeelden zullen we gebruik maken van de module *turtle* waarmee we eenvoudige tekeningen kunnen maken. Op deze manier kunnen we de verschillende constructies visualiseren. Met de *turtle* module kunnen we een virtuele schildpad besturen; met verschillende commando's kunnen we de schildpad vooruit laten gaan, links en rechtsom draaien, een pen laten neerzetten, ... Concreet kunnen we volgende commando's gebruiken:

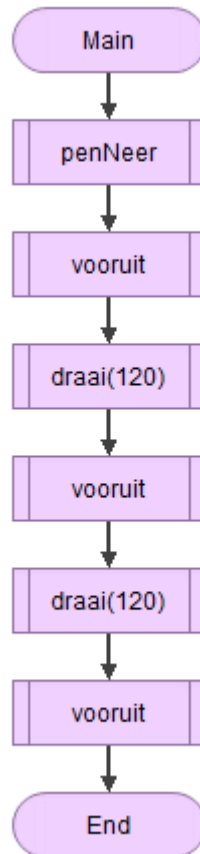
- *reset()* : maak het canvas leeg en zorgt dat de schildpad op haar beginpositie staat;
- *forward(100)* : ga 100 stappen vooruit. Uiteraard kan je 100 door elk ander getal vervangen;
- *left(45)* : draai 45 graden naar links;
- *right(45)* : draai 45 graden naar rechts;
- *penup()* : hef de pen op; de schildpad laat geen spoor na;
- *pendown()* : zet de pen neer; als de schildpad beweegt, laat ze een spoor na;
- *goto(100,100)* : de schildpad beweegt in een rechte lijn naar positie (100,100.) De schildpad start in het midden van het canvas op positie (0,0). De linker benedenhoek heeft coördinaat (-200,-200), de rechterbovenhoek (200,200). De schildpad "kijkt naar rechts"; met andere woorden: *forward(50)* zal de schildpad 50 stappen naar rechts laten bewegen.

Om de *turtle* module te kunnen gebruiken, moeten we eerst deze module importeren. Dit kan met het volgende commando. **Voer de volgende cel uit voordat je de volgende cellen uitvoert; anders zullen die niet correct werken.** Je doet dit door de cel te selecteren en op *Run* te drukken in de toolbar.

```
In [1]: from turtle import *
```

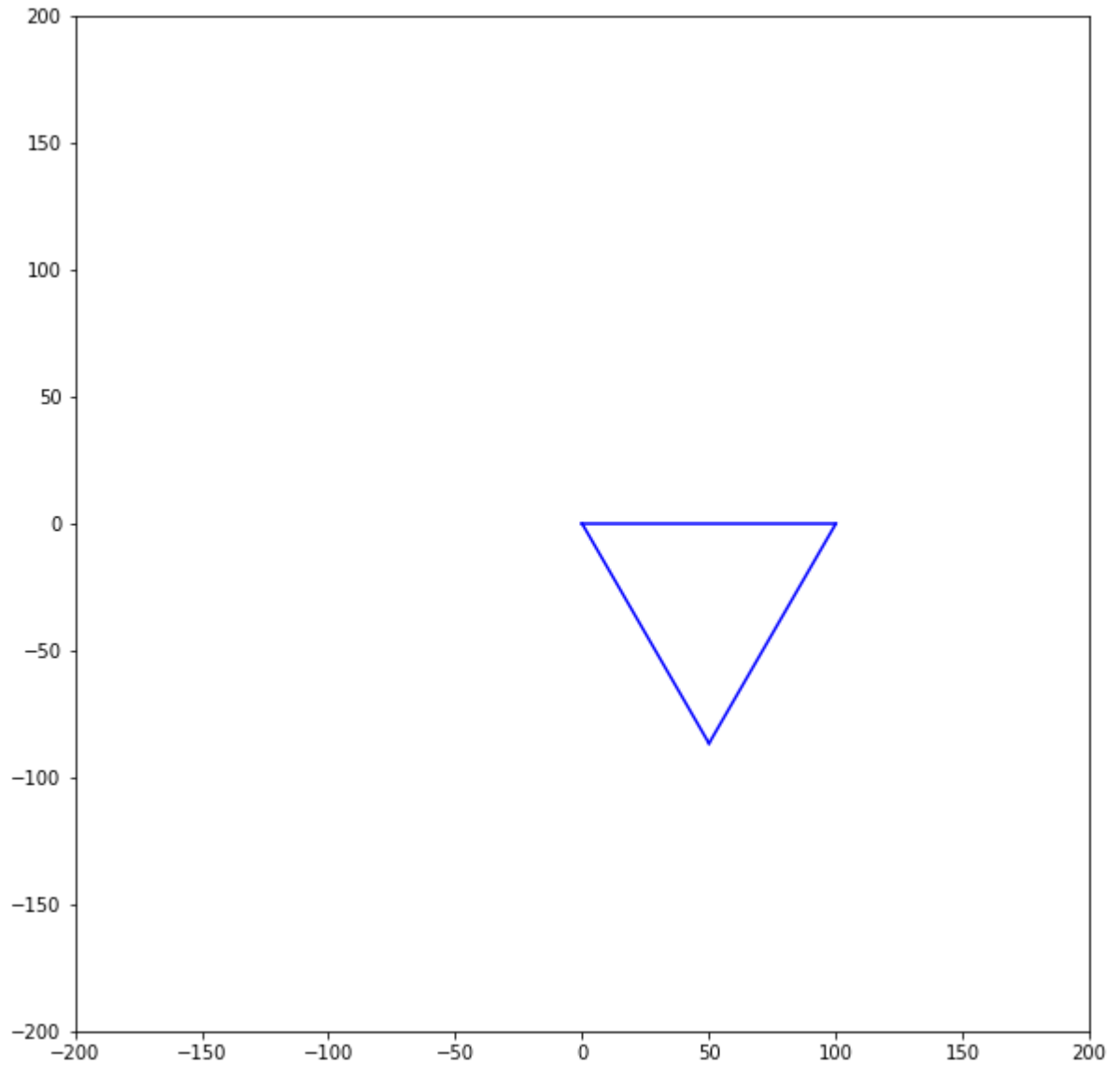
Sequentiële uitvoering

De eerste en eenvoudigste constructie is de sequentiële uitvoering van commando's. Als we een driehoek willen tekenen, kunnen we gebruik maken van volgende sequentie van stappen:



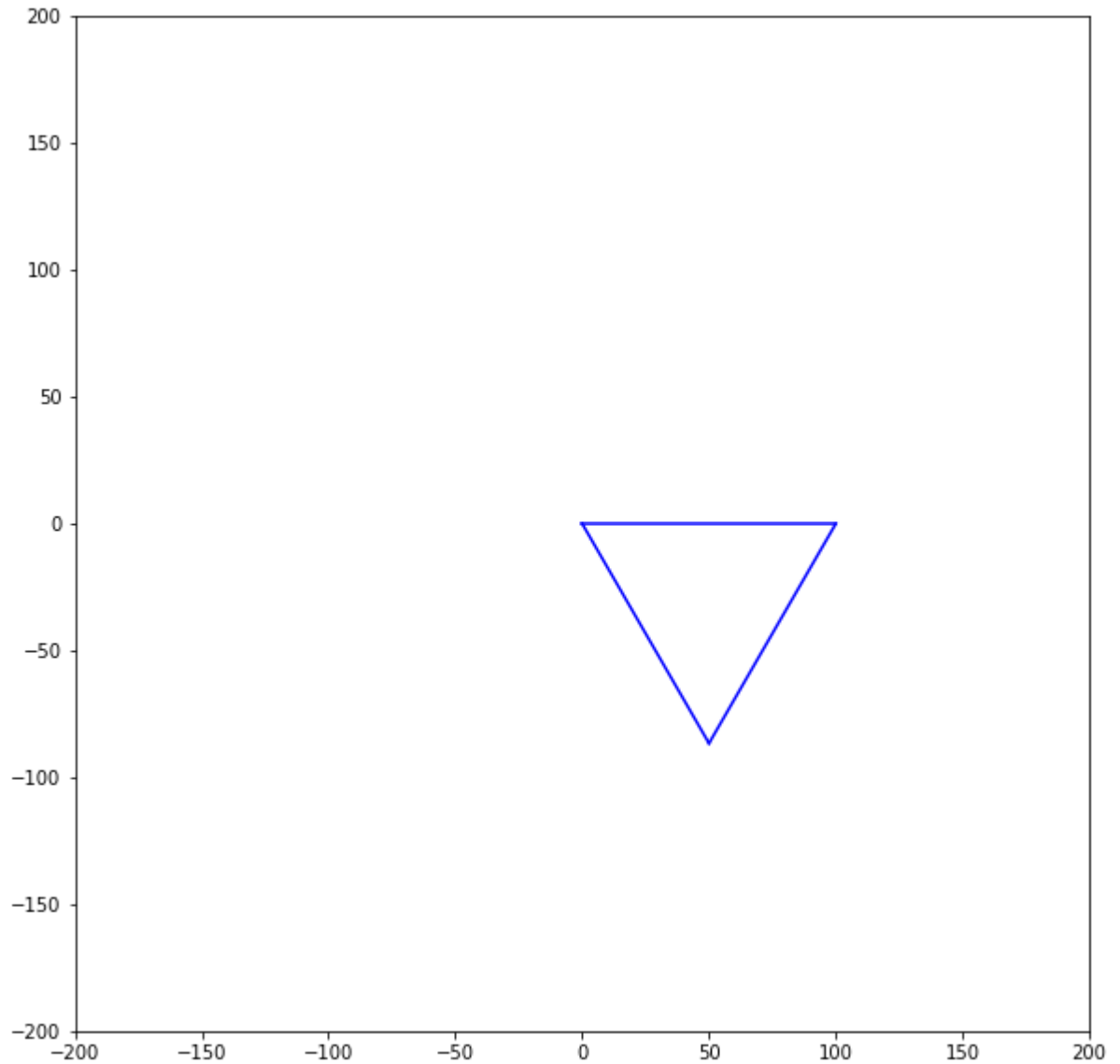
Commando's die achter elkaar geplaatst worden, worden een voor een uitgevoerd. Lege regels zijn toegestaan, maar alle commando's moeten in het begin van de regel beginnen; niet voorafgegaan door tabs of spaties dus. Volgende reeks commando's tekent een driehoek:

```
In [2]: reset()  
  
forward(100)  
right(120)  
forward(100)  
right(120)  
forward(100)
```



Pas nu deze code aan om een vierkant af te drukken:

```
In [3]: reset()  
  
forward(100)  
right(120)  
forward(100)  
right(120)  
forward(100)
```

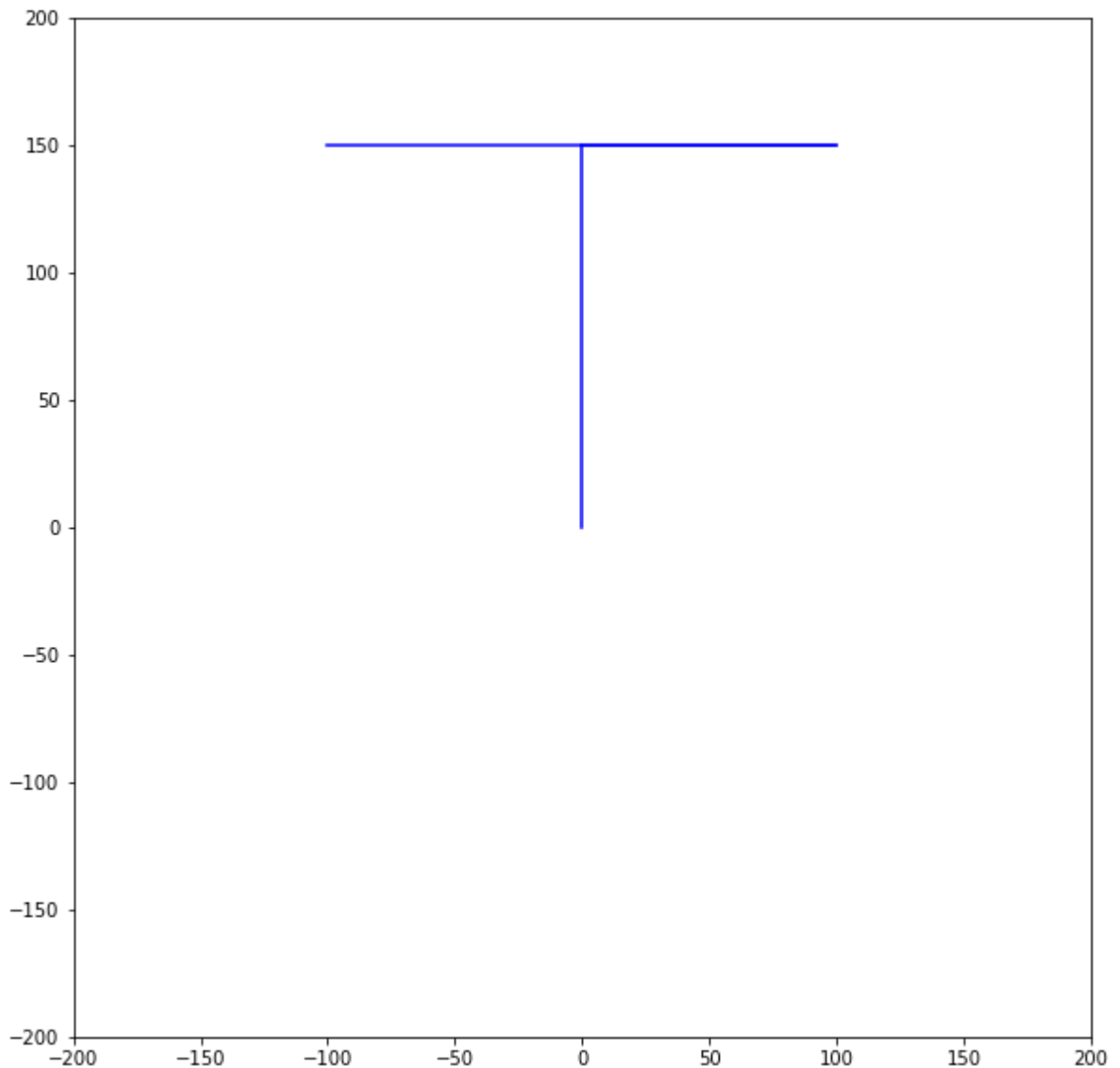


Met behulp van het karakter # kunnen we commentaar opnemen om de code meer leesbaar te maken. De *Python interpreter* (het programma dat de code leest en uitvoert) zal alles wat volgt na het karakter # negeren:

```
In [4]: # Tweede opdracht: pas volgende code aan zodat het de eerste letter van jouw n
        # aam tekent.
        # Tim, Tina, Toon, Tessa, ... tekenen de eerste letter van hun achternaam!

        reset() # Anders staat de turtle nog zoals na het vorige programma

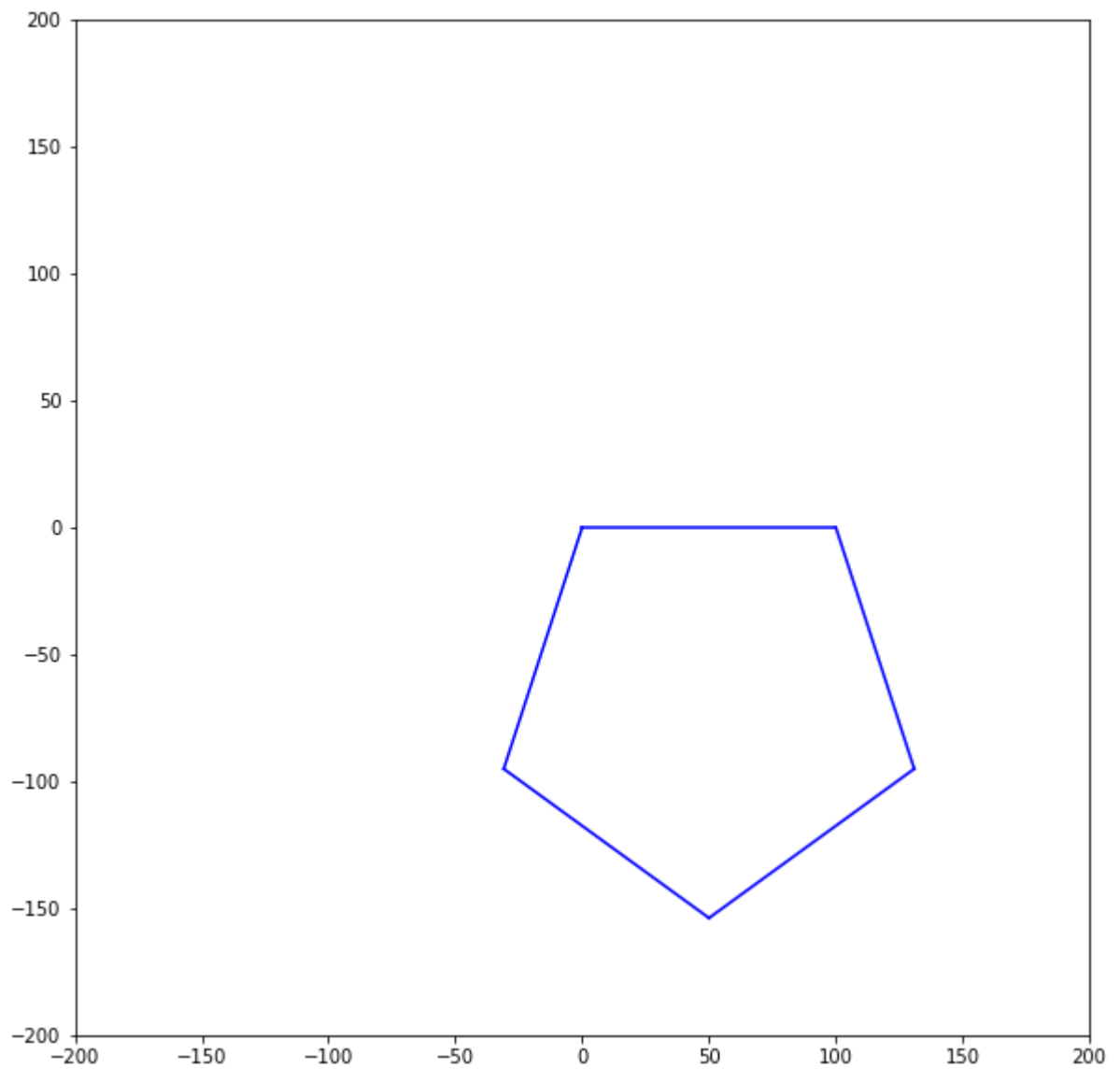
        penup()
        goto(-100,150)
        pendown()
        forward(200)
        left(180)
        forward(100)
        left(90)
        forward(150)
```



Herhalingslus - *while-loop*

Vaak moeten we in code dezelfde stappen meermaals herhalen. Bijvoorbeeld: stel dat we een vijfhoek willen tekenen. Dan kunnen we dit doen als volgt:

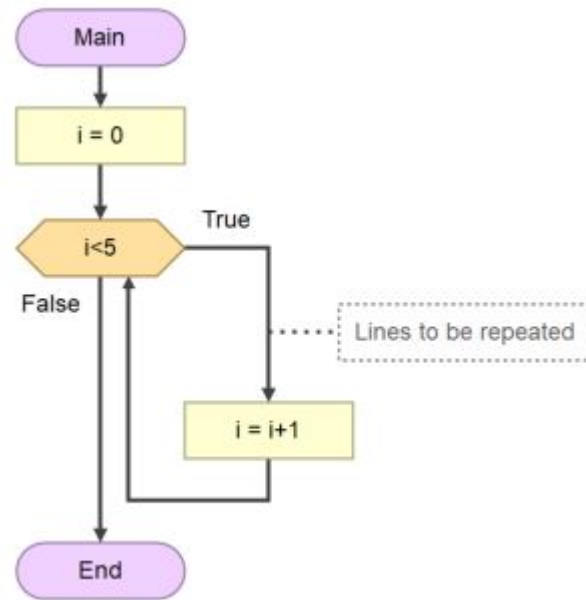
```
In [5]: reset()  
forward(100)  
right(72)  
forward(100)  
right(72)  
forward(100)  
right(72)  
forward(100)  
right(72)  
forward(100)  
right(72)
```



We herhalen hierbij de volgende 2 regels 5 maal:

`forward(100) right(72)`

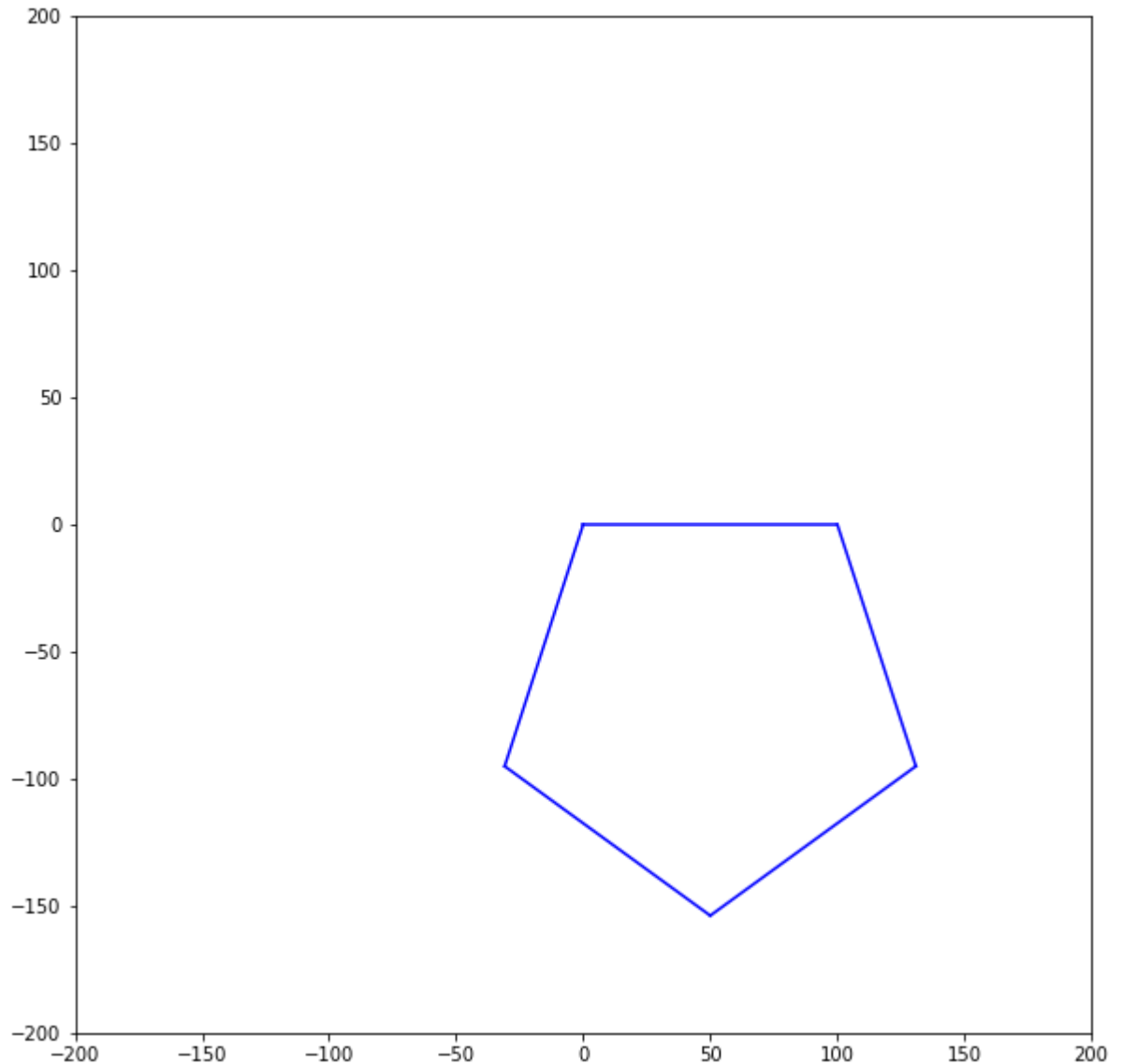
Dat kan makkelijker met een zogenaamde *loop*:



In Python kunnen we dit implementeren met een *while-loop*. (Later zien we ook nog de zogenaamde *for-loop*)

```
In [6]: reset()
```

```
i=0  
while i<5:  
    forward(100)  
    right(72)  
    i=i+1
```



while i<5: betekent hier: herhaal zolang de *variabele i* kleiner is dan 5. We zullen later in deze notebook ingaan op de exacte betekenis van de *variabele i* die hier wordt gebruikt, maar voorlopig kan je aan *i* denken als een tellertje dat bijhoudt hoe vaak we al herhaald hebben. Voor de *while* wordt *i* op 0 gezet; we hebben het stukje code nog niet (dus 0 maal) herhaald. Vervolgens voeren we de 2 lijnen code uit (om vooruit te gaan en te draaien) en verhogen we het tellertje met 1 via het commando *i=i+1*.

Pas onderstaande code nu aan zodat er een 10-hoek wordt getekend; je moet hiervoor de vraagtekens vervangen door getallen:


```
In [7]: # Pas onderstaande code aan. Op de plaatsen waar een vraagteken staat moet je
        # een waarde invullen

        reset()

        penup()      # Verplaats de schildpad eerst zodat die niet buiten het canvas g
        #aat bij het tekenen.
        goto(-50,150)
        pendown()

        i=0
        while i<?:
            forward(100)
            right(?)
            i=i+1

File "<ipython-input-7-bd2d7166223b>", line 10
    while i<?:
            ^
SyntaxError: invalid syntax
```

Merk op dat de regels die herhaald moeten worden in de *while-loop* geïndenteerd zijn. Dat wil zeggen: ze springen in. Gebruik voor de insprong steeds 4 spaties, niet meer, niet minder. Vermijd ook het gebruik van tabs. Een goede editor zal normaal gezien tabs automatisch vervangen door 4 spaties. Ook springen de meeste editors automatisch 4 spaties in als de vorige regel op een dubbele punt (":") eindigde. Als het blok dat herhaald moet worden gedaan is, spring je terug 4 spaties naar links.

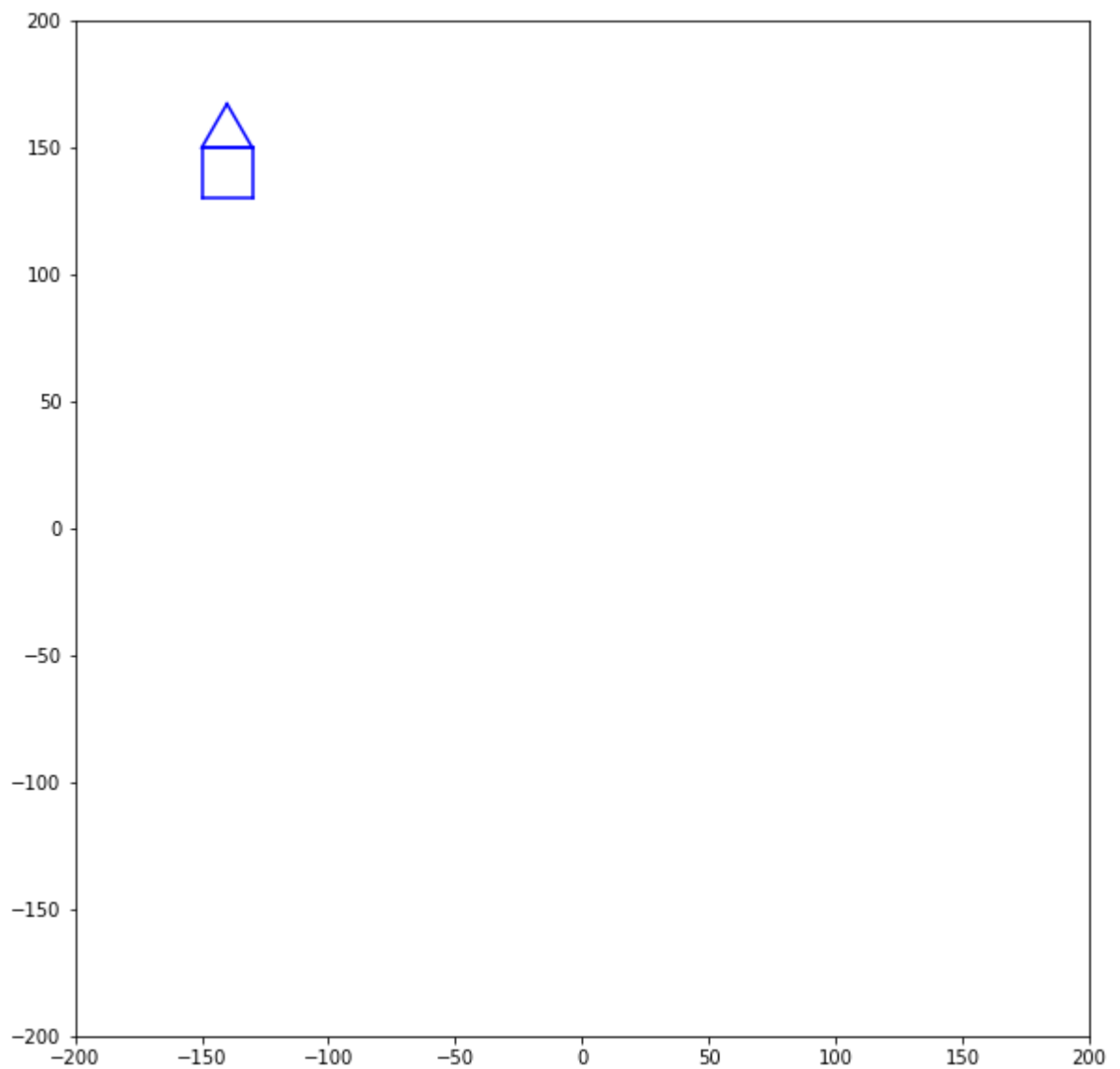
Je kan verschillende *while-loops* combineren en zelfs *nesten*. "Nesten" betekent dat je een *while-loop* binnen een andere *while-loop* plaatst. We geven nu enkele voorbeelden waarbij eerst enkele *while-loops* na elkaar worden gebruikt en nadien enkele *while-loops* worden "genest".

Het eerste voorbeeld tekent een klein huisje door eerst een vierkant te tekenen en daarna een driehoek:

```
In [8]: reset()

penup()
goto(-150,150)
pendown()

i=0
while i<4:
    forward(20)
    right(90)
    i=i+1
i=0
while i<3:
    forward(20)
    left(120)
    i=i+1
```



Volgende code herhaalt dit twee maal met een verplaatsing daartussen om twee huisjes te tekenen:

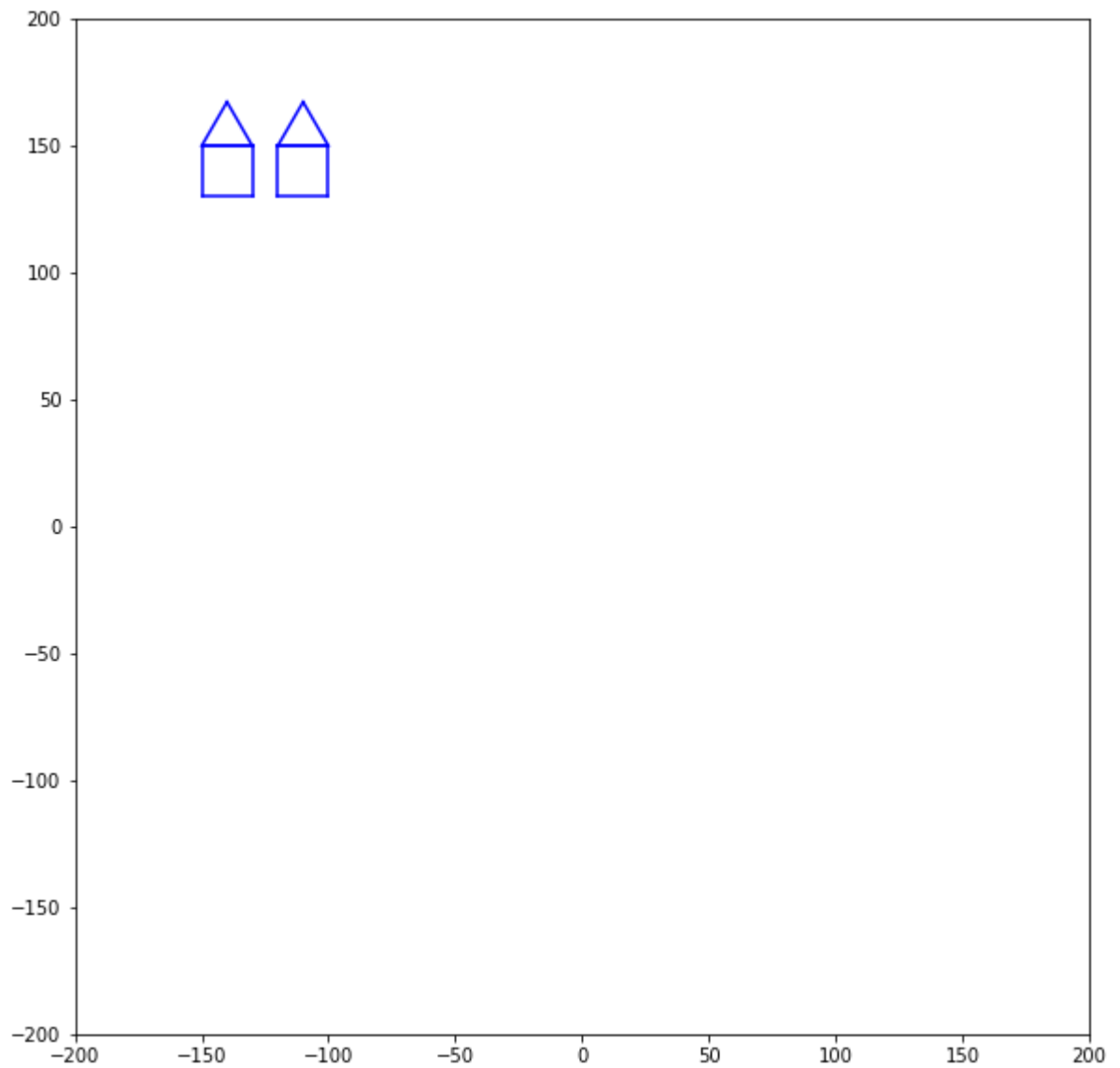
```
In [9]: reset()

penup()
goto(-150,150)
pendown()

i=0
while i<4:
    forward(20)
    right(90)
    i=i+1
i=0
while i<3:
    forward(20)
    left(120)
    i=i+1

penup()    # verplaats de schildpad naar rechts zodat de huisjes niet boven op
elkaar getekend worden
forward(30)
pendown()

i=0
while i<4:
    forward(20)
    right(90)
    i=i+1
i=0
while i<3:
    forward(20)
    left(120)
    i=i+1
```



We kunnen dit echter ook met een *while-loop* doen. Merk op dat we hierbij wel een andere *variabele* moeten gebruiken; we gebruiken voor onze buitenste *while-loop* de variabele *j*:

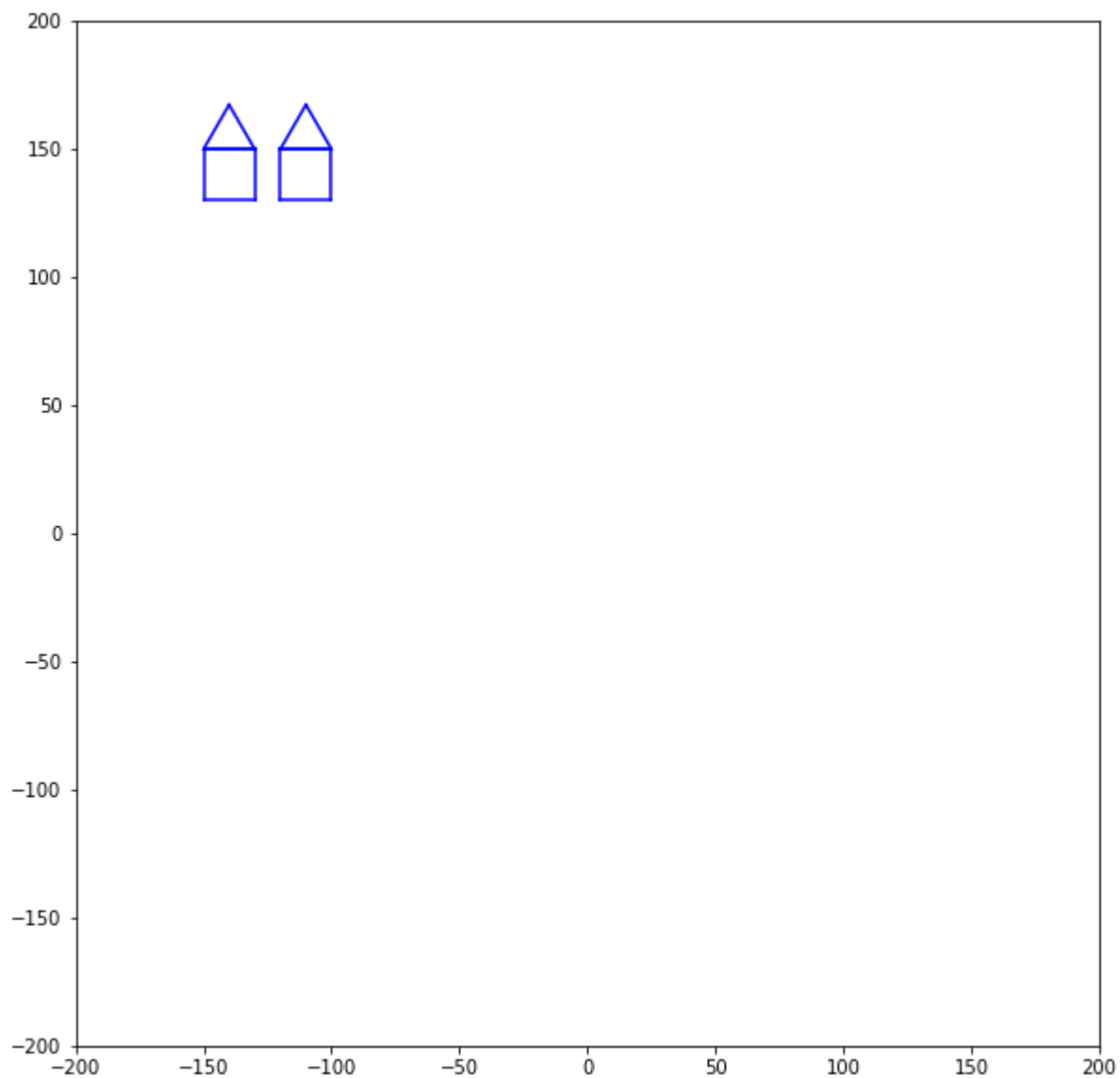
```
In [10]: reset()

penup()
goto(-150,150)
pendown()

j=0
while j<2:
    i=0
    while i<4:
        forward(20)
        right(90)
        i=i+1
    i=0
    while i<3:
        forward(20)
        left(120)
        i=i+1

    penup()
    forward(30)
    pendown()

    j=j+1
```



Zo kunnen we ook 10 huisjes tekenen:

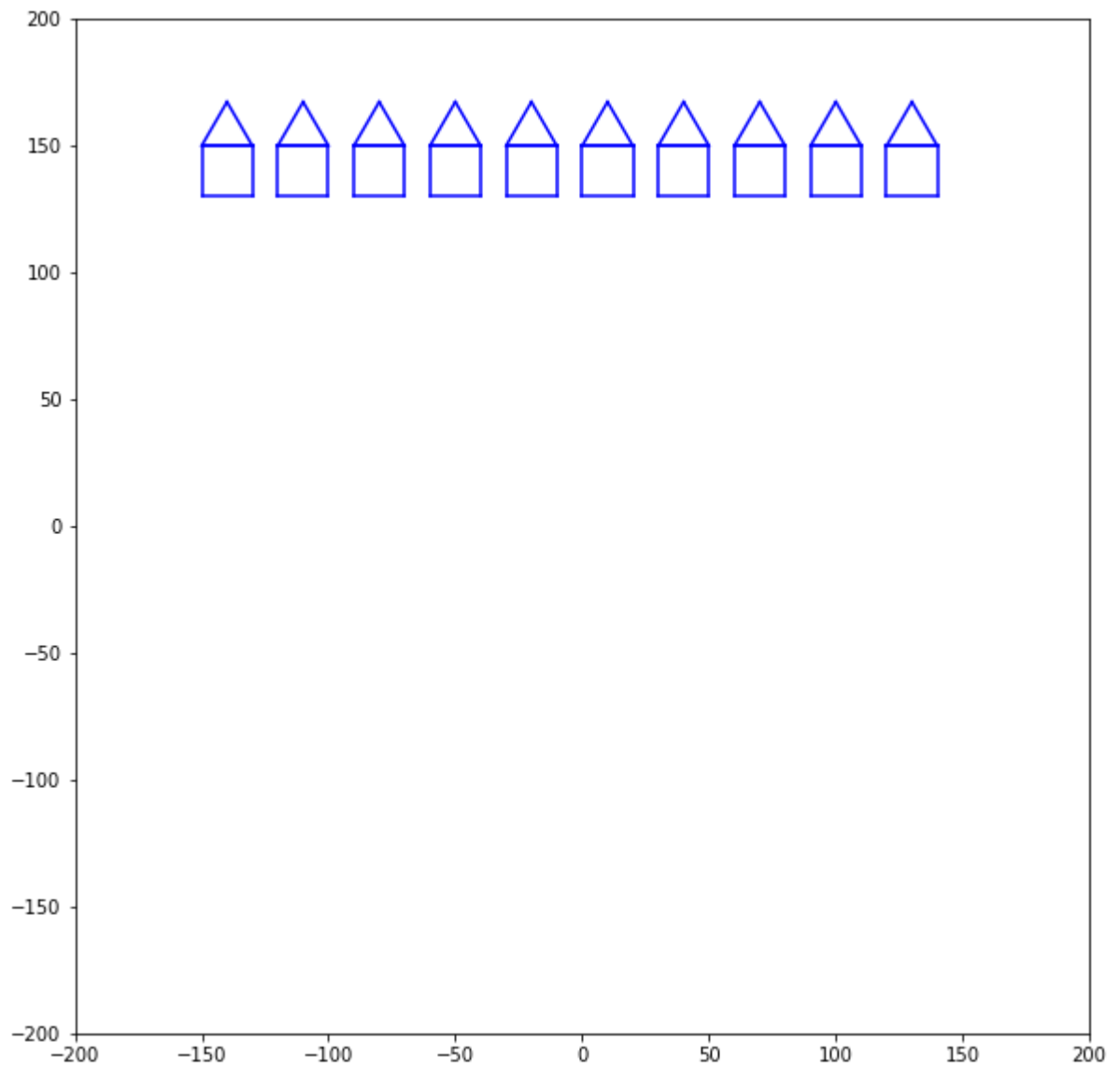
```
In [11]: reset()

penup()
goto(-150,150)
pendown()

j=0
while j<10:
    i=0
    while i<4:
        forward(20)
        right(90)
        i=i+1
    i=0
    while i<3:
        forward(20)
        left(120)
        i=i+1

    penup()
    forward(30)
    pendown()

    j=j+1
```



Twee rijen huizen? Geen probleem! We herhalen de code gewoon twee maal!


```
In [12]: reset()

penup()
goto(-150,150)
pendown()

j=0
while j<10:
    i=0
    while i<4:
        forward(20)
        right(90)
        i=i+1
    i=0
    while i<3:
        forward(20)
        left(120)
        i=i+1

    penup()
    forward(30)
    pendown()

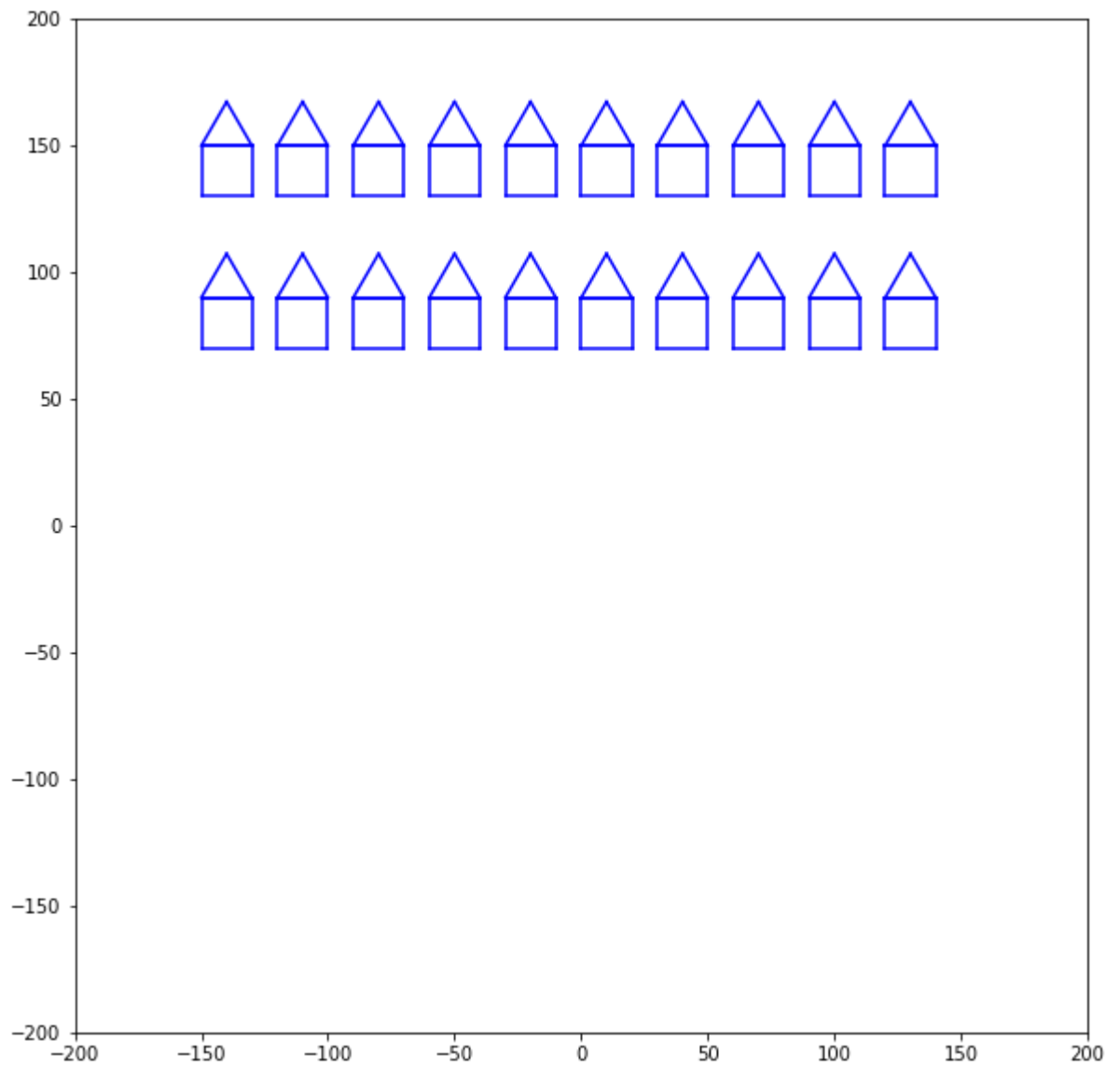
    j=j+1

penup()
right(90)    # Ga naar onder
forward(60)
right(90)    # En daarna terug helemaal naar links
forward(300)
right(180)
pendown()

j=0
while j<10:
    i=0
    while i<4:
        forward(20)
        right(90)
        i=i+1
    i=0
    while i<3:
        forward(20)
        left(120)
        i=i+1

    penup()
    forward(30)
    pendown()

    j=j+1
```



Je raadt het nooit; we kunnen dit ook met nog een extra "nesting" van een *for-loop*:

```
In [13]: reset()

penup()
goto(-150,150)
pendown()

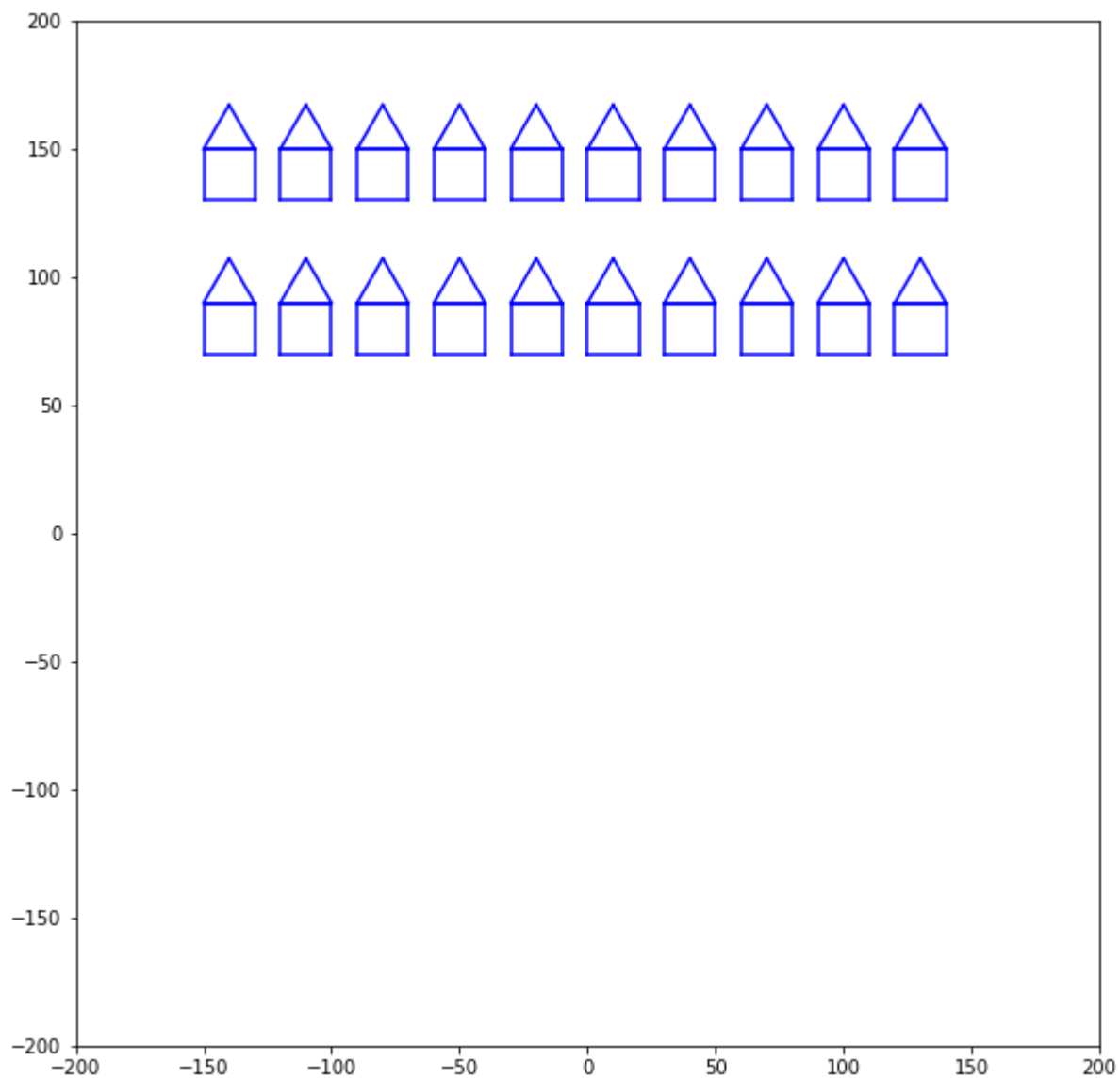
k=0
while k<2:
    j=0
    while j<10:
        i=0
        while i<4:
            forward(20)
            right(90)
            i=i+1
        i=0
        while i<3:
            forward(20)
            left(120)
            i=i+1

        penup()
        forward(30)
        pendown()

        j=j+1

    penup()
    right(90)
    forward(60)
    right(90)
    forward(300)
    right(180)
    pendown()

    k=k+1
```



En dus ook makkelijk 5 rijen huizen:

```
In [14]: reset()

penup()
goto(-150,150)
pendown()

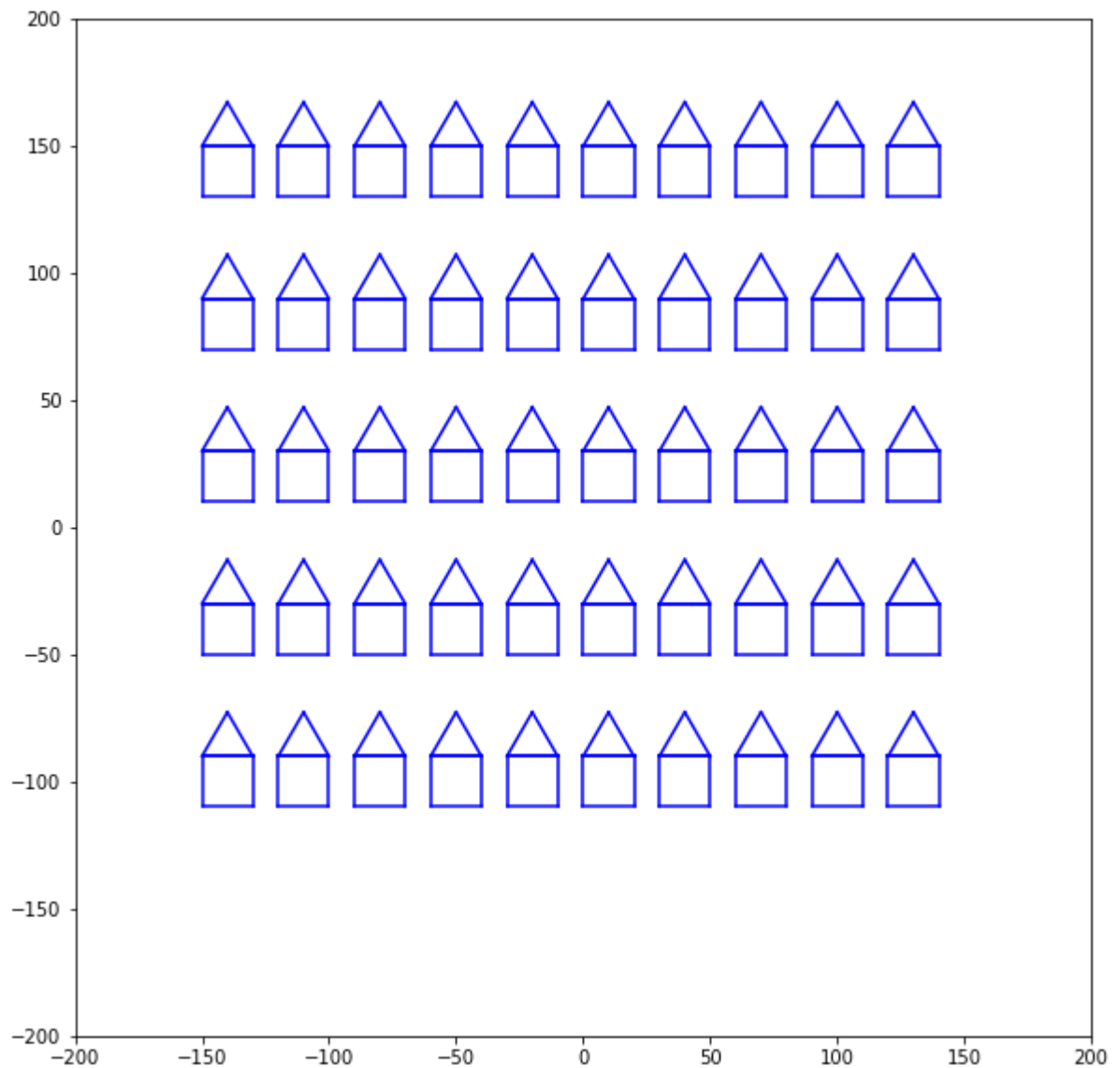
k=0
while k<5:
    j=0
    while j<10:
        i=0
        while i<4:
            forward(20)
            right(90)
            i=i+1
        i=0
        while i<3:
            forward(20)
            left(120)
            i=i+1

        penup()
        forward(30)
        pendown()

        j=j+1

    penup()
    right(90)
    forward(60)
    right(90)
    forward(300)
    right(180)
    pendown()

    k=k+1
```



Variabelen

De i , j , k die we in bij de *while-loop* gebruikten zijn zogenaamde *variabelen*. Variabelen zijn grootheden die we zelf definiëren, waar we waarden aan kunnen toekennen, de waarde van kunnen aanpassen en mee kunnen rekenen. We zouden bijvoorbeeld een variabele N kunnen in gebruik nemen waarin we het aantal hoeken opslaan als we een regelmatige N-hoek willen tekenen:

```
In [15]: N = 5 # aantal hoeken
```

Dit noemen we een *toekenning* of *assignment*; we kennen waarde 5 toe aan variabele *N*. Later in het programma kunnen we *N* gebruiken op alle plaatsen waar een waarde verwacht wordt. In tegenstelling tot in vele andere programmeertalen hoef je in Python niet op voorhand een lijst variabelen te declareren, maar kan je als je een variabele nodig hebt, gewoon een waarde toekennen. Een variabele naam kan bestaan uit meerdere letters, een combinatie van letters en cijfers, underscores ("_"), maar kan niet starten met een cijfer. Verder zijn er een aantal gereserveerde namen, zoals "for".

Nu we het aantal hoeken weten, kunnen we berekenen over hoeveel graden we moeten draaien om een regelmatige N-hoek te maken:

```
In [16]: hoek = 360/N
```

"/" staat voor deling. Andere *operatoren* zijn onder andere "*" (vermenigvuldiging), "+", "-", "%" (modulo = rest bij deling).

Om te controleren wat de waardes zijn van *N* en *hoek* zijn, kunnen we de functie *print* gebruiken:

```
In [17]: print(N)
         print(hoek)
```

```
5
72.0
```

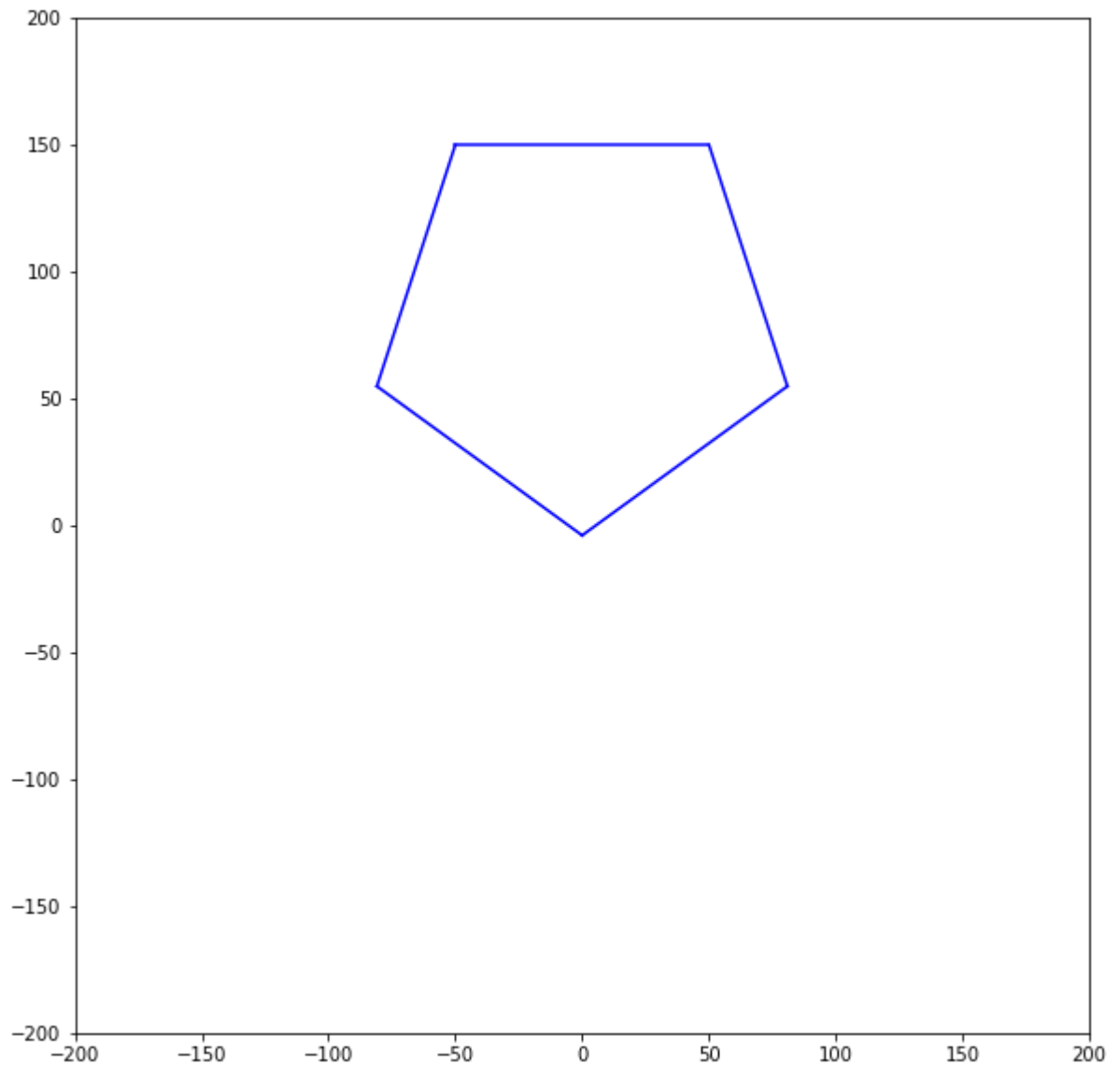
Nu kunnen we *N* en *hoek* gebruiken in ons programma:

```
In [18]: N=5
hoek=360/N

reset()

penup()      # Verplaats de schildpad eerst zodat die niet buiten het canvas g
aat bij het tekenen.
goto(-50,150)
pendown()

i=0
while i<N:
    forward(100)
    right(hoek)
    i=i+1
```



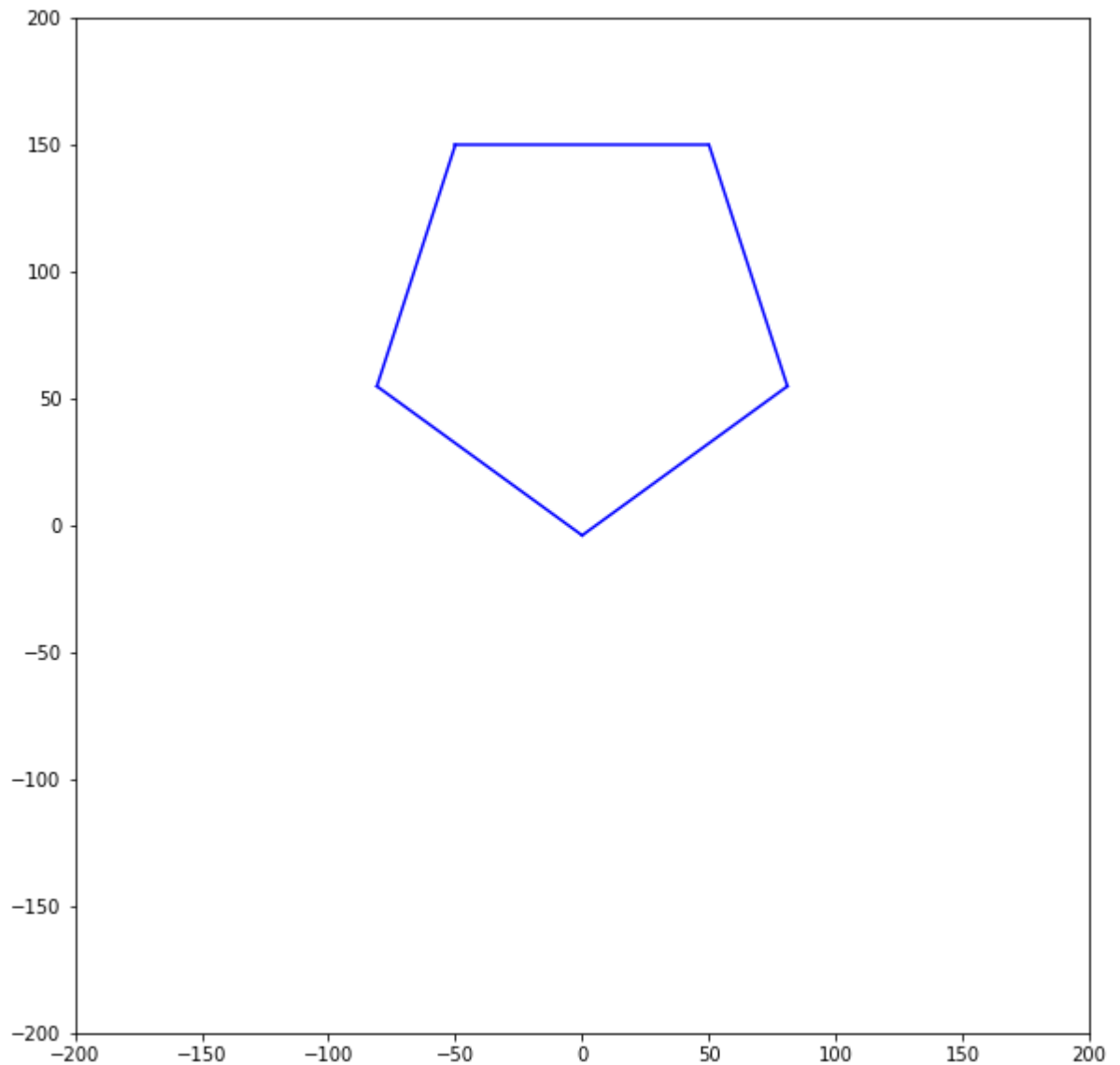
Merk op dat als we N te groot maken, we buiten het canvas tekenen. Pas onderstaande code aan zodat de lijstukken in plaats van steeds lengte 100, kleiner worden al naargelang N groter wordt:


```
In [19]: N=5
hoek=360/N

reset()

penup()      # Verplaats de schildpad eerst zodat die niet buiten het canvas gaat bij het tekenen.
goto(-50,150)
pendown()

i=0
while i<N:
    forward(100) # op deze regel moet je iets veranderen
    right(hoek)
    i=i+1
```

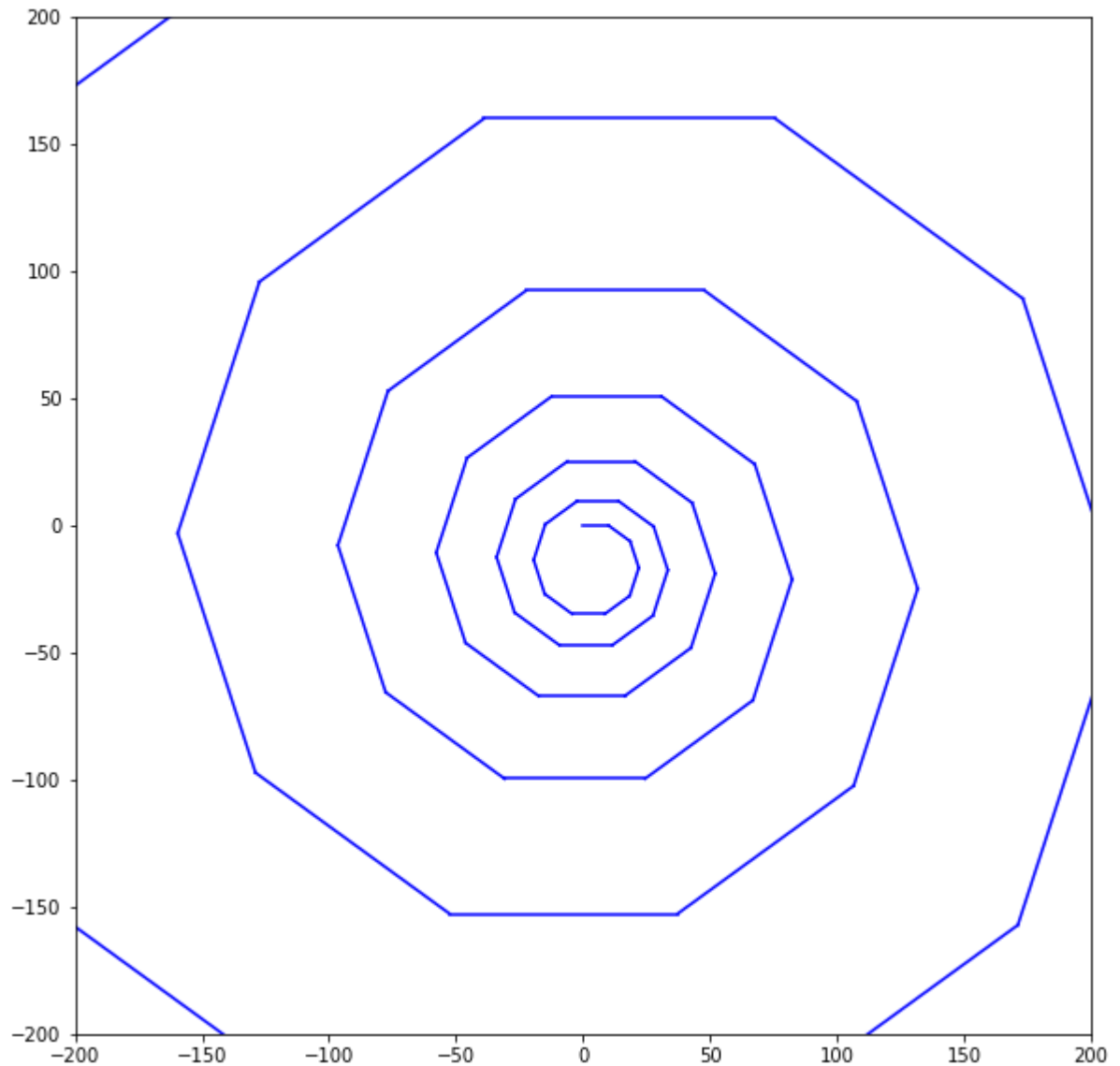


Volgend stukje code tekent een naar buiten draaiend spiraal door de lengte van het lijnstuk steeds te vergroten:

```
In [20]: N=100
hoek=36
lengte=10

reset()

i=0
while i<N:
    forward(lengte)
    right(hoek)
    lengte=lengte*1.05 # Merk op: een komma-getal schrijven we met een .
    i=i+1
```



Conditionele uitvoering: if - elif - else

Soms moeten we in een programma een keuze maken en een stukje code *conditioneel* uitvoeren. Dat kan met de *if* in Python. De *syntax* is als volgt:

if [conditie]: blok dat enkel wordt uitgevoerd als conditie waar is

Zo tekent volgend blokje code opnieuw rijen huisjes, met het verschil dat in de i -de rij het i -de huisje niet wordt getekend. Hiervoor gebruiken we de conditie $k \neq j$. " $\neq$ " betekent "zijn ongelijk". Aangezien k het nummer van de rij is dat wordt getekend en j het nummer van het huisje, is $k \neq j$ waar indien het huisje dat getekend moet worden niet hetzelfde nummer heeft als de rij waarin het getekend wordt. De code voor het tekenen van het huisje springt vervolgens in, en wordt dus enkel uitgevoerd als aan de conditie $k \neq j$ is voldaan.

```
In [21]: reset()

penup()
goto(-150,150)
pendown()

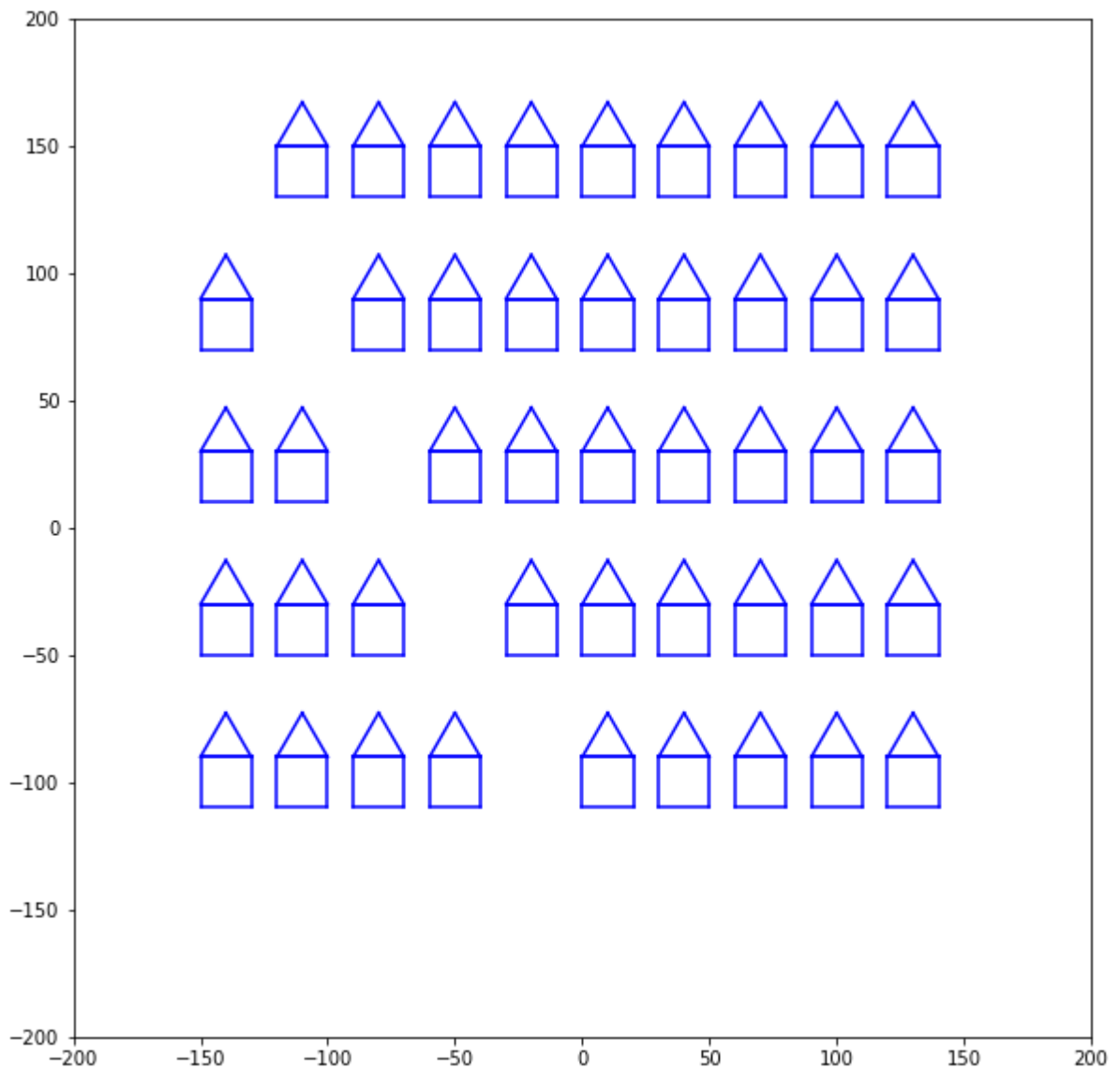
k=0
while k<5:
    j=0
    while j<10:
        i=0
        if k!=j:
            while i<4:
                forward(20)
                right(90)
                i=i+1
            i=0
            while i<3:
                forward(20)
                left(120)
                i=i+1

        penup()
        forward(30)
        pendown()

        j=j+1

    penup()
    right(90)
    forward(60)
    right(90)
    forward(300)
    right(180)
    pendown()

    k=k+1
```



In het voorbeeld hierboven waren er twee mogelijkheden:

- ofwel is de conditie $k \neq j$ voldaan en de code om het huisje te tekenen wordt uitgevoerd;
- ofwel is de conditie niet voldaan en wordt de code niet uitgevoerd.

In Python zijn er echter nog meer mogelijkheden. Zo kunnen we een cascade van condities maken waarbij de code die hoort bij de eerste conditie waaraan voldaan is, wordt uitgevoerd. Dat doen we met *if-elif-else*, met volgende syntax:

if [conditie1]: blok dat enkel wordt uitgevoerd als conditie1 waar is elif [conditie2]: blok dat enkel wordt uitgevoerd als conditie1 NIET waar is, en conditie2 wel waar is elif [conditie3]: blok dat enkel wordt uitgevoerd als conditie1 NIET waar is, conditie2 NIET waar is en conditie3 wel waar is else: blok dat enkel wordt uitgevoerd als conditie1 NIET waar is, conditie2 NIET waar is en conditie3 NIET waar is

Merk op dat we een onbeperkt aantal takken kunnen maken in de beslissingsboom die het programma volgt, maar dat er slechts 1 zal worden uitgevoerd. Als er een *else* is dan zal er steeds exact 1 stukje code worden uitgevoerd, aangezien het blokje code bij de *else* wordt uitgevoerd als geen enkele andere conditie voldaan is. Merk ook op dat we de condities van boven naar onder aflopen en enkel het blokje code uitvoeren dat hoort bij de eerste conditie waaraan voldaan is. Als er aan meerdere condities voldaan is, wordt dus enkel het stukje code uitgevoerd dat geassocieerd is met de eerste conditie waaraan voldaan is.

Volgende code illustreert dit; opnieuw tekenen we het k -de huisje in de k -de rij niet. Verder tekenen we bij het derde huisje (het huisje met nummer $j=2$) geen dak. Merk op dat `==` betekent "is gelijk aan". We moeten hier een dubbele `=="` gebruiken omdat Python anders in de war raakt met de toekenning $j=2$.

```

In [22]: reset()

penup()
goto(-150,150)
pendown()

k=0
while k<5:
    j=0
    while j<10:
        i=0
        if j==2: # derde huisje krijgt geen dak
            while i<4:
                forward(20)
                right(90)
                i=i+1
            i=0
        elif j!=k: # op de k-de rij tekenen we huisje *k* niet
            while i<4:
                forward(20)
                right(90)
                i=i+1
            i=0
            while i<3:
                forward(20)
                left(120)
                i=i+1

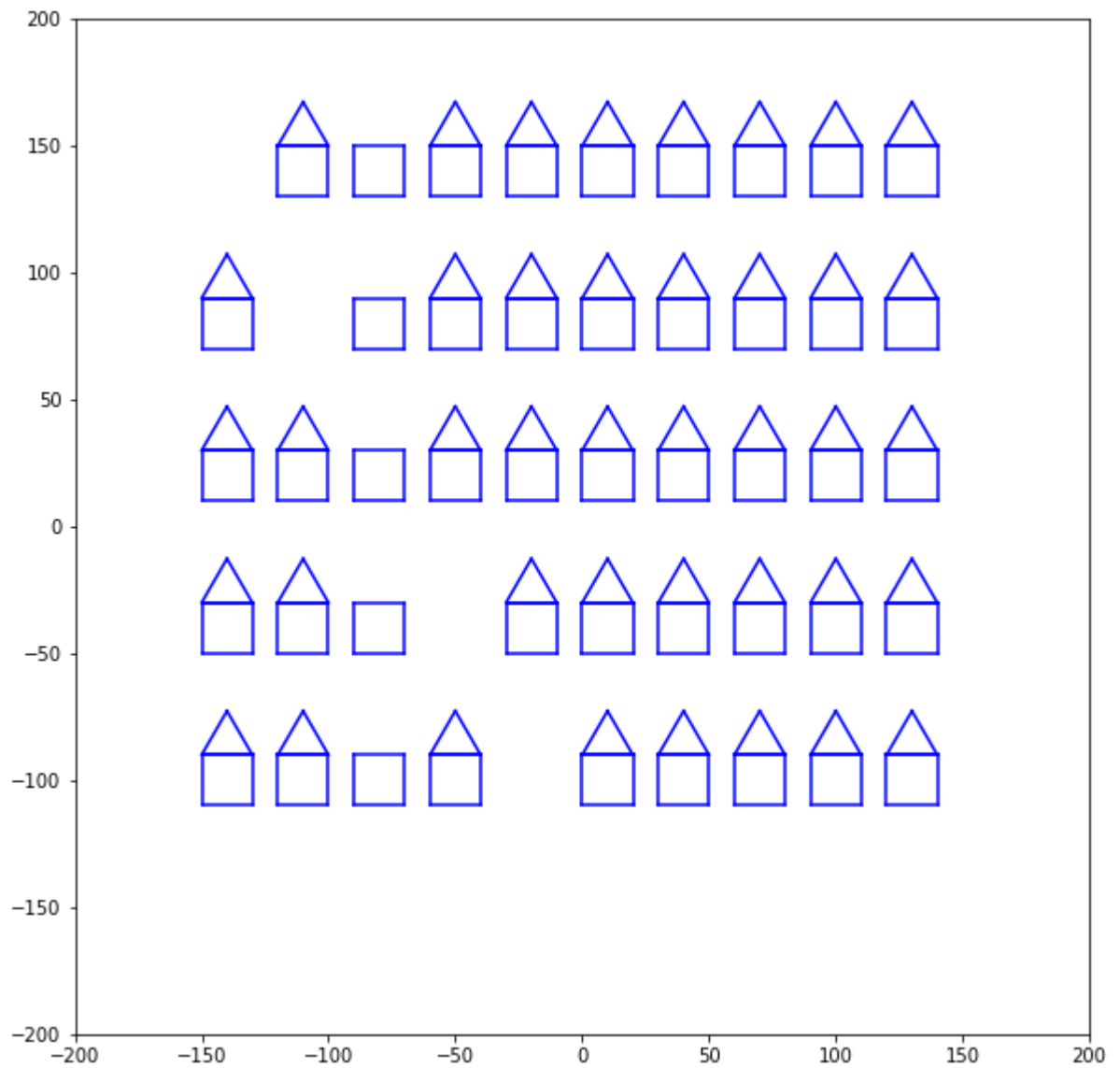
        penup()
        forward(30)
        pendown()

        j=j+1

    penup()
    right(90)
    forward(60)
    right(90)
    forward(300)
    right(180)
    pendown()

    k=k+1

```



Merk op dat we in de derde rij wel het derde huisje tekenen, zonder dak, omdat hier de conditie $j==2$ is voldaan en we dus de code uitvoeren die daarbij hoort. Volgende code met een *geneste if* tekent ook in de derde rij het derde huisje niet. Bekijk de code; begrijp je waarom niet?


```
In [23]: reset()

penup()
goto(-150,150)
pendown()

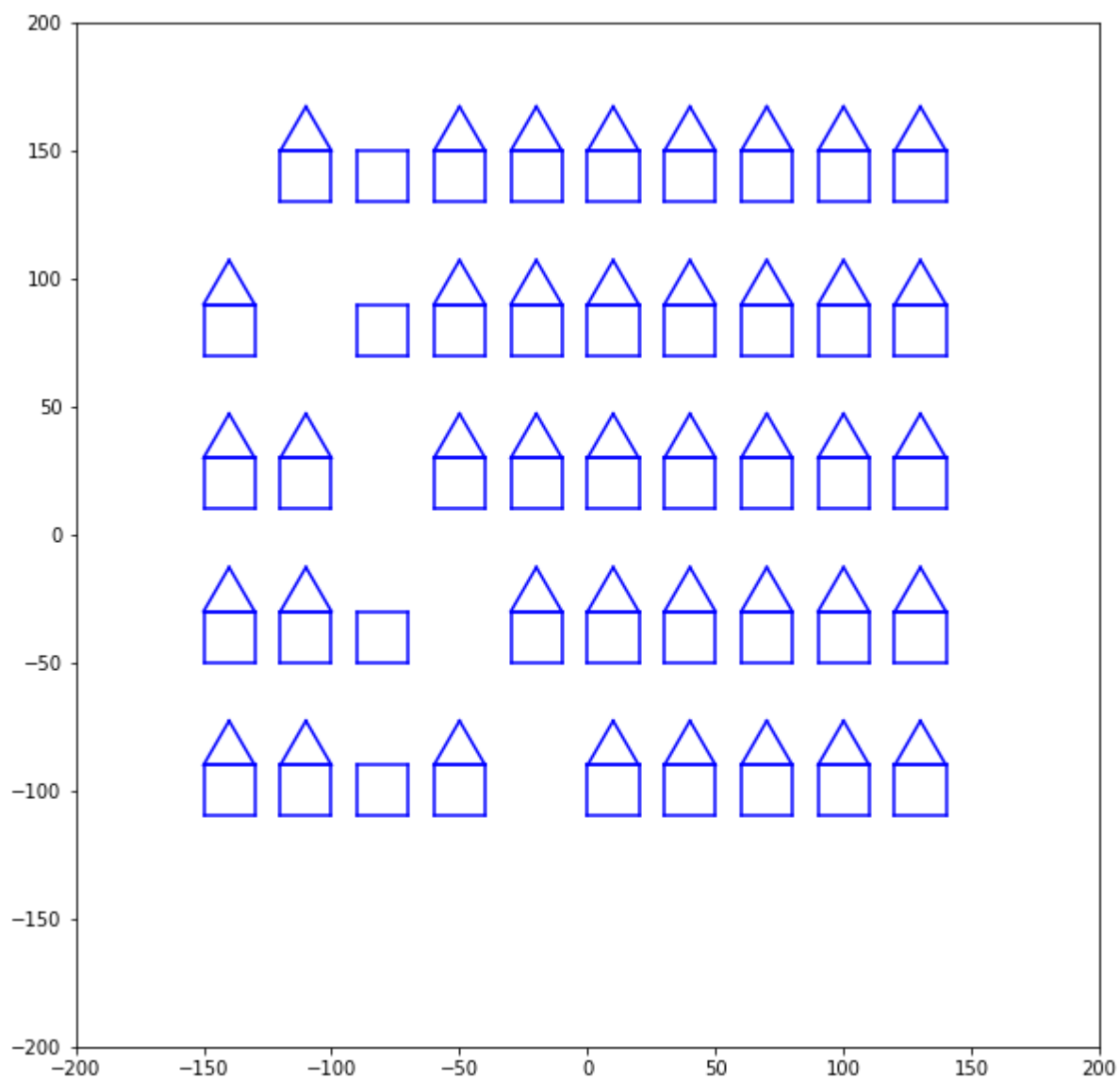
k=0
while k<5:
    j=0
    while j<10:
        i=0
        if j!=k: # op de k-de rij tekenen we huisje *k* niet
            while i<4:
                forward(20)
                right(90)
                i=i+1
            i=0
            if j!=2: # het derde huisje krijgt geen dak
                while i<3:
                    forward(20)
                    left(120)
                    i=i+1

        penup()
        forward(30)
        pendown()

        j=j+1

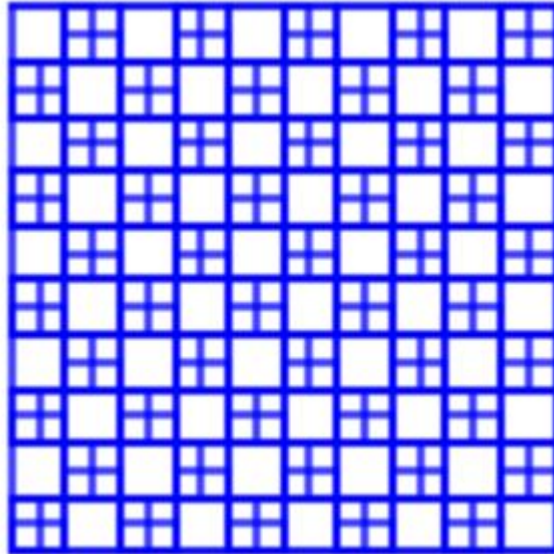
    penup()
    right(90)
    forward(60)
    right(90)
    forward(300)
    right(180)
    pendown()

    k=k+1
```



Extra uitgewerkt voorbeeld

Geen nieuwe constructies meer, maar een uitgewerkt voorbeeld. Het is de bedoeling om volgende figuur te tekenen:



Denk goed na hoe je het tekenen van eze figuur opsplittst in logische blokjes (hier letterlijk te nemen). Bekijk daarna de code. Komt dit overeen met jouw idee?

```
In [24]: reset()

# Ga eerst naar de Linkerbovenhoek
penup()
goto(-180,180)
pendown()

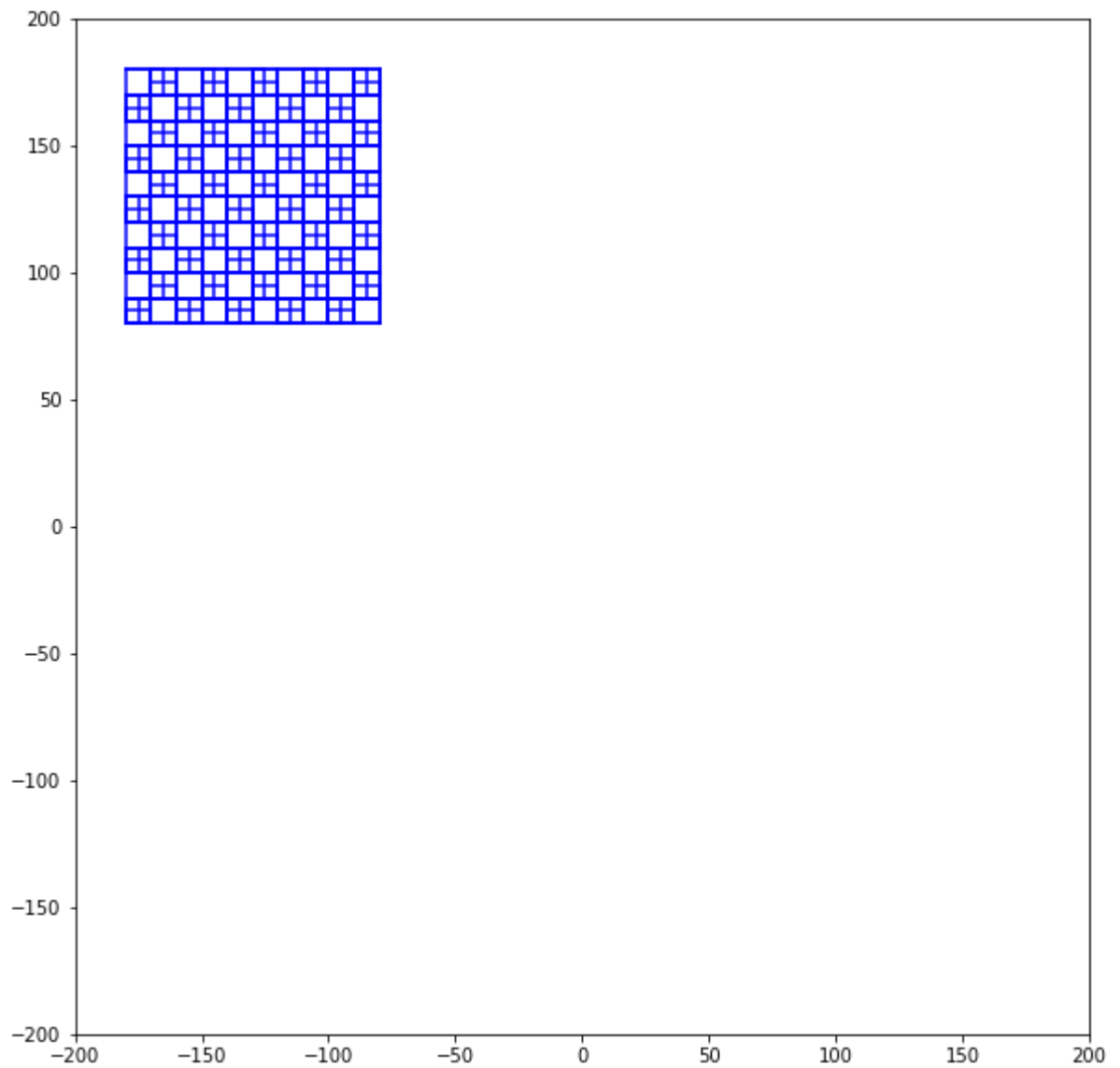
leeg=1

k=0
while k<10:
    j=0
    while j<10:
        # lege blok:
        i=0
        while i<4:
            forward(10)
            right(90)
            i=i+1
        if leeg!=1: # Blok met +
            forward(5)
            right(90)
            forward(10)
            right(90)
            forward(5)
            right(90)
            forward(5)
            right(90)
            forward(10)
            right(90)
            forward(5)
            right(90)
            forward(10)
            right(90)
            forward(10)
            right(90)
            forward(10)
            right(90)

        forward(10) # schuif een plaatsje op naar rechts
        j=j+1 # verhoog het kolomnummer
        if j!=10: # als we niet iop het einde van de rij zijn
            if leeg==1: # Als deze blok leeg was moet de volgende vol zijn
                leeg=0
            else: # en omgekeerd
                leeg=1

        # verplaats naar volgende rij
        right(90)
        forward(10)
        right(90)
        forward(100)
        right(180)

    k=k+1 # verhoog het rijnummer
```



Conclusie:

In deze notebook zagen jullie reeds een eerste set van basisconstructies van de Python programmeertaal:

- sequentie
- *while*-loop
- conditionele uitvoering met *if*
- variabelen

Het is belangrijk om deze constructies grondig in te oefenen tijdens de practica.