Quickguide in Python

# Syntax, variables & data types

### Syntax & whitespace

* Whitespace is heel belangrijk in python. Plaats geen whitespaces waar dit niet nodig is, aangezien whitespace een deel van de syntax is.
* Indentatie wordt gedaan met 4 spaties.

### Variables & data types

* Je hoeft **geen** data type te geven aan een variable, dit gebeurt automatisch.
* Een value van een variable kan gereassigned worden door de variable opnieuw te vullen met een nieuwe waarde.

|  |  |
| --- | --- |
| **Code** | **Result** |
| print X | Prints X |
| variable\_name = X | Plaatst X in variable\_name  Dit kan eender welk data type zijn |
| #Single line comment | Alles na de # wordt gecomment |
| “””Multi-Line Comments  komen tussen  tripple quotations  “”” | Alles tussen de “”” (tripple quotations) wordt gecomment |
| my\_string = “dit is een string” | Data type: string  Value: dit is een string |
| my\_int = 23 | Data type: integer  Value: 23 |
| my\_float = 1.25 | Data type: float  Value: 1.25 |
| my\_bool = True | Data type: boolean  Value: True  Note: “True & False horen met een hoofdletter geschreven te worden” |

Voorbeeldcode:

melk = “Volle melk”

hoeveelheid = 3

print melk

print hoeveelheid

Result:

Volle Melk

3

# Math

### Math & operators

* Wiskunde in programmeren wordt gedaan met operators. Operators zijn “+ - \* /”. Hierop zijn nog enkele variaties zoals modulo en macht.
* Gebruik een of meerdere wiskundige values, zet er een operator tussen en je krijgt een wiskundige bewerking.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Result** | **Syntax** | **Result** |
| + | Adition (optellen) | addition = 158 + 322 | 488 |
| - | Subtraction (aftrekken) | subtraction = 680 - 144 | 536 |
| \* | Multiplication (vermenigvuldiging) | multiplication = 51 \* 86 | 4386 |
| / | Division (deling) | division = 125 / 25 | 5 |
| \*\* | Power (macht) | power = 2 \*\* 3 | 8 |
| % | Modulo (modulo, rest) | modulo = 3%2 | 1 |

Voorbeeldcode:

meal = 44.50

tax = 0.0675

tip = 0.15

meal = meal + meal \* tax

total = meal + meal \* tip

print total

Result:

54.63

# Strings

### Basics

Een string is de letterlijke waarde van de inhoud. Een string “17” is exact dat, de letters “1” en “7”. Strings hebben geen wiskundige inhoud en hier kan men dan als gevolg ook niet mee rekenen. Bijvoorbeeld: “17” + “17” wordt “1717” en niet “34”.

### Concatinatie

Met dit voorbeeld hebben we ook ineens concatenatie gezien. Concatenatie is het samenvoegen van verschillende waarde. Dit kan met alle datatypes. Dit kan gedaan worden door reeds bestaande variables en nieuwe strings.

Voorbeeldcode:

aantal\_glazen = 25

print “Aantal glazen op de kast: “ + aantal\_glazen + “ stuks.”

Result:

Aantal glazen op de kast: 25 stuks.

### Escape

Maar wat nu als we het volgende willen uitschrijven: Ik wil het boek genaamd “Lord of the Rings” kopen.

Het probleem dat we hier hebben is dat we de string sluiten net voor het wordt Lord.

Om dit op te lossen gebruiken we escapes. Het escape symbool in Python is “\”. De eerste letter of het eerste symbool dat je hiervoor zet wordt letterlijk geschreven en niet gezien als syntax. De code ziet er dan zo uit.

Voorbeeldcode:

print “Ik wil het boek genaamd \”Lord of the Rings\” kopen.

### Access by Index

Elke letter in een string heeft een nummer. Dit noemen we de index. De index begint te tellen vanaf 0. Als we bijvoorbeeld de 5e letter willen hebben van een string gaan we als volgt tewerk, rekening houdende dat we bij 0 beginnen te tellen.

Voorbeeldcode:

fifth\_letter = “Een string”

print fifth\_letter[4]

Result:

S

### String methods

String methods laten je bepaalde acties uitvoeren op een string.

|  |  |
| --- | --- |
| String Method | Function |
| len() | Geeft de lengte van een string terug |
| lower() | Zet heel de string om in lower case |
| upper() | Zet heel de string om in upper case |
| str() | Maakt strings van niet string variables  (dus maakt bijvoorbeeld een string van een float) |

Voorbeeldcode 1:

parrot = "Norwegian Blue"

print len(parrot)

Result:

14

Voorbeeldcode 2 (andere schrijfwijze, doet hetzelfde):

parrot = "Norwegian Blue"

print parrot.len()

Result:

14

### String formatting

Je kan placeholders gebruiken die vervangen kunnen worden door strings. We gebruiken de “%” operator om de “%s” placeholders te vervangen door variables. Je moet dezelfde hoeveelheid “%s” placeholders hebben als het aantal variables. De syntax ziet er als volgt uit:

Voorbeeldcode:

name = raw\_input("What is your name?")

quest = raw\_input("What is your quest?")

color = raw\_input("What is your favorite color?")

print "Ah, so your name is %s, your quest is %s, " \

"and your favorite color is %s." % (name, quest, color)

Note:

raw\_input() vraagt de user om input via een prompt met een vraag en plaats de gegeven value in een variable.

### Slicing

Je kan een woord slicen door te zeggen van welk karakter tot welk karakter hij moet ‘slicen’.

Voorbeeldcode:

string\_variable[2:9]

# Date & Time

### Datetime

Soms is het nodig om te weten wanneer we iets gedaan hebben of om andere bewerkingen te doen met tijd. Om te kunnen werken met tijd moeten we eerst de module inladen. Dat gaat met de volgende lijn code “from datetime import datetime”. Als we dit bovenaan van onze code zetten zal de module ingeladen worden en kunnen we ermee beginnen werken.

|  |  |
| --- | --- |
| **Code** | **Result** |
| from datetime import date | Import datetime module |
| datetime.now() | De datum en tijd op het punt van declaratie |

Voorbeeldcode:

from datetime import datetime

now = datetime.now()

Eens we de datetime in een variable hebben gestoken kunnen we details eruit vissen.

|  |  |
| --- | --- |
| Code | Result |
| now.year | Geeft het jaar terug van de datetime variable now |
| now.month | Geeft de maand terug van de datetime variable now |
| now.day | Geeft de dag terug van de datetime variable now |
| now.hour | Geeft het uur terug van de datetime variable now |
| now.minute | Geeft de minuut terug van de datetime variable now |
| now.second | Geeft de second terug van de datetime variable now |

# Conditionals & Control Flow

### Comparators

Soms willen we dat code beslissingen voor ons maakt. Dit kan aan de hand van comparators.

|  |  |
| --- | --- |
| **Code** | **Result** |
| == | Is gelijk aan |
| != | Is niet gelijk aan |
| < | Is kleiner/minder dan |
| <= | Is kleiner/minder OF gelijk aan |
| > | Is groter/meer dan |
| >= | Is groter/meer OF gelijk aan |

### Boolean operators

Boolean operators vergelijken statements en resultaten in boolean waardes, er zijn 3 boolean operators

|  |  |
| --- | --- |
| **Code** | **Result** |
| and | Kijkt of beide statements kloppen |
| or | Kijkt minstens één van de waardes kloppen |
| not | Geeft het tegenovergestelde van een statement, dus kijkt of iets niet klopt |

### Conditional Statements

“if” is een conditional statement dat een bepaalde blok code gaat uitvoeren als de statement “True” is.

Voorbeeldcode:

if 125 == 25 \* 5:

print “Output text”

#### Syntax

Merk op dat het resultaat intenderen heel belangrijk is. De “if” functie gaat de code uitvoeren dat ingesprongen is. Ook belangrijk is het dubbelpunt op het einde van je if statement. Dit gaat ook tellen voor de “else” statement dat hierop volgt.

De “else” en “elif” statement zorgt ervoor dat de code wordt uitgevoerd bij de else block als de if “false” teruggeeft.

Voorbeeldcode:

if 125 == 25 \* 3:

print “Deze statement klopt niet dus deze blok code wordt overgeslagen”

elif 125 == 25 \* 4:

print “Deze statement klopt ook niet dus hij gaat verder”

else:

print “De if statement klopt niet dus deze blok wordt in de plaats uitgevoerd”

# Functions

Soms willen we stukken code hergebruiken. In plaats van alles opnieuw te schrijven is het veel properder als we dit eenmalig in een blok code zouden kunnen zetten en hergebruiken. Hiervoor gebruiken we functions. Een functie bestaat uit 3 componenten: header, de optionele comment en de body.

###### Header

De header begint met de “def” tag en wordt gevolgd door de parameters die je wil meegeven. Parameters zijn values/variables die van buiten je functie komen maar naar je functie worden gestuurd om er eventueel iets mee te doen. De volgorde is hierin belangrijk, want als je de functie oproept gaat hij de waardes steken in de variabelnamen in de volgorde waarin ze staan.

Syntaxvoorbeeld: “def (variable\_naam1, variable\_naam2)”

###### (Optionele) Comment

De comment dient om te zeggen wat de functie doet.

Syntaxvoorbeeld: “”””Return getal van 1 tot 500”””

###### Body

Dit is je eigenlijke functie. Hierin ga je de code schrijven dat je wil hergebruiken. Belangrijk: gebruik 1 intendation voor je body.

Voorbeeldcode:

def hello\_person(person\_name):

print “Hello “ + person\_name

hello\_person(“Mark”)

Result:

Hello Mark

Een functie kan een andere functie oproepen op dezelfde manier dat je normaal een functie oproept.

Voorbeeldcode:

def one\_good\_turn(n):

return n + 1

def deserves\_another(n):

return one\_good\_turn(n) + 2

###### Math Module

Python heeft een hoop wiskundige functies ingebouwd. Hiervoor gebruiken we een module genaamd “math”. Om de module te kunnen gebruiken moet je hem eerst importeren door “import math” te schrijven voor je “math” gebruikt.

Nadat math geïmporteerd kunnen volgende functies gebruikt worden als je “math.” ervoor zet, dus bvb “math.sqrt(25)”.

|  |  |
| --- | --- |
| **Code** | **Result** |
| math.sqrt(x) | Geeft de vierkantswortel van x |

Het is ook mogenlijk om 1 functie van een module te importeren met “from module import function”. Bijvoorbeeld “from math import sqrt”. Als je dit doet moet je math. niet meer gebruiken en kan je bijvoorbeeld gewoon schrijven sqrt(x). Om alles te importeren van een module kan je “\*” gebruiken. Dan wordt het “from math import \*”.

Universele imports lijkt een goed idee maar heeft als nadeel dat het je programma vol steekt met variabelen. Daarom ik een standard “import module” beter, zo moet je “modulename.functionname” gebruiken en blijven jou variable namen veilig.

Om alle functies te zien van een module kan je het volgende doen:

Voorbeeldcode:

import math # importeert math module

everything = dir(math) # zet alles in een lijst van “math”

print everything # print ze allemaal uit

###### Ingebouwde Modules

Python heeft een hoop ingebouwde modules die we niet moeten importeren. Hiervan zijn we er al een paar eerder tegengekomen (zie: “String Methods”). Hieronder nog enkele nuttige voorbeelden.

|  |  |
| --- | --- |
| **Code** | **Result** |
| max(x, y, z) | Return de hoogste waarde van een aantal argumenten |
| min(x, y, z) | Return de laagste waarde van een aantal argumenten |
| abs(x) | Return de absolute value van X. Dit wil zeggen de afstand van 0. Het geeft altijd een positief getal terug en neemt maar 1 argument aan |
| type(x) | Return het variable type van x |

# Lists & Dictionaries

###### Lists

Lists zijn een datatype dat verschillende stukken informatie opslaan onder één variablenaam. Je kan een lijst declareren als volgt: “list\_name = [item\_1, item\_2]”.  
Je kan ook een lege lijst declareren: “empty\_list = []”. Lists beginnen te tellen vanaf 0, dus als je de 5e positie wil oproepen moet je “list\_name[4]”.

###### Append

Om gemakkelijk een item achteraan toe te voegen kan je de append functie gebruiken.

Voorbeeldcode:

suitcase = []

suitcase.append("sunglasses")

suitcase.append("boat")

suitcase.append("car")

suitcase.append("something")

list\_length = len(suitcase) # Set this to the length of suitcase

print "There are %d items in the suitcase." % (list\_length)

print suitcase

Result:

There are 4 items in the suitcase.

['sunglasses', 'boat', 'car', 'something']

###### List Slicing

Soms wil je slechts een deel oproepen van een list. Dit bijvoorbeeld als volgt:

Voorbeeldcode:

suitcase = ["sunglasses", "hat", "passport", "laptop", "suit", "shoes"]

# The first and second items (index zero and one)

first = suitcase[0:2]

# Third and fourth items (index two and three)

middle = suitcase[2:4]

# The last two items (index four and five)

last = suitcase[len(suitcase)-2:len(suitcase)]

Om te slicen vanaf een stuk en alles ervoor of erna te nemen hoor je de gewenste kant blanco te houden: suitcase[:13] neemt alle waardes voor positie 12 (want je begint te tellen vanaf 0).

###### List functions

|  |  |
| --- | --- |
| **Code** | **Result** |
| variablenaam.index(“string”) | Geeft de index terug als hij een value “string” terugvind in de list |
| variablenaam.insert(1, “string”) | Voegt de value “string” toe aan de list op positie 1 en schuift alle andere values 1 positie op |
| variablenaam.sort() | Sorteert alle items in variablenaam alfabetisch |

Voorbeeldcode:

animals = ["aardvark", "badger", "duck", "emu", "fennec fox"]

duck\_index = animals.index("duck")

animals.insert(duck\_index, "cobra")

print animals

Result

['aardvark', 'badger', 'cobra', 'duck', 'emu', 'fennec fox']

###### For Loops

Als je voor elk item in een lijst iets wil uitvoeren ga je een for loop gebruiken.

Voorbeeldcode:

my\_list = [1,9,3,8,5,7]

for number in my\_list:

# Your code here

print 2\*number

Result:

2

18

6

16

10

14

De for doorloopt alle items van my\_list. Telkens als hij bij een nieuwe value komt steekt hij die value in “number” zodat je er bewerkingen mee kunt doen. In dit geval vermenigvuldigt hij elk getal in de lijst met 2.

###### Dictionaries

Je kan ook werken met keys in plaats van een index. Hierbij geeft je in de plaats van een index een naam mee aan een value. Dit noemen we dan dictionaries

Voorbeeldcode:

residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}

print residents['Puffin']

print residents['Sloth']

print residents['Burmese Python']

Result:

104

105

106

###### Nieuwe Entries & delete entries

Een nieuwe entry invullen is heel simpel: “dict\_name[new\_key] = new\_value”, als je een bestaande key aanspreekt zal hij hem vervangen met de nieuwe value dat je meegeeft. Om een value te verwijderen doen we: “del dict\_name[key\_name]”.

###### Even Herhalen

Al de dingen die we net leerden kunnen we door elkaar gebruiken. Hieronder vind je wat code dat demonstreert hoe je bijvoorbeeld een inventory zou kunnen bijhouden voor een game.

Voorbeeldcode:

inventory = {

'gold' : 500,

'pouch' : ['flint', 'twine', 'gemstone'], # Assigned a new list to 'pouch' key

'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']

}

inventory['burlap bag'] = ['apple', 'small ruby', 'three-toed sloth']

inventory['pouch'].sort()

inventory['pocket'] = ['seashell', 'strange berry', 'lint']

inventory['backpack'].sort()

inventory['backpack'].remove('dagger')

inventory['gold'] += 50

# Loops

###### While Loop

De while loop is gelijkaardig aan een if statement. Het herhaalt een blok code zolang een statement klopt.

Voorbeeldcode:

number = 0

while number < 5

codeblok

number += 1

###### While / else

Iets eigen aan Python is de while/else loop. Deze trekt op if/else maar is net iets anders.De else blok wordt elke keer uitgevoerd wanneer de loop condition false is. Dit wil zeggen dat wilt zeggen dat het else gedeelte wordt uitgevoerd als de loop nooit begint of wanneer hij normaal eindigt, dus niet als je hem breaked. De else wordt maar 1 maal uitgevoerd.

###### Break

Je kan eender welke loop breken door een “break” te zetten in je code. Als je loop voorbij een break komt verbreekt hij de lus en gaat verder met de code na de loop.