Python, een korte introductie (voor C# ontwikkelaars)

Python is een ontwikkeltaal die al enige tijd sterk in opkomst is. Als 'T-shaped' software ontwikkelaar kunnen we hier niet langer om heen. Dit is voor ons ook de aanleiding geweest om eens een blik te werpen op deze taal. In dit artikel willen we een globale introductie geven van Python de ontwikkeltaal en daarnaast de kenmerkende verschillen met C# aanstippen.

Hoe is het ontstaan

"Python is een programmeertaal die begin jaren 90 ontworpen en ontwikkeld werd door Guido van Rossum, destijds verbonden aan het Centrum voor Wiskunde en Informatica (daarvoor Mathematisch Centrum) in Amsterdam. De taal is mede gebaseerd op inzichten van professor Lambert Meertens, die een taal genaamd ABC had ontworpen, bedoeld als alternatief voor BASIC, maar dan met geavanceerde datastructuren. Inmiddels wordt de taal doorontwikkeld door een enthousiaste groep, tot juli 2018 geleid door Van Rossum. Deze groep wordt ondersteund door vrijwilligers op het internet. De ontwikkeling van Python wordt geleid door de Python Software Foundation. Python is vrije software.
Python heeft zijn naam te danken aan het favoriete televisieprogramma van Guido van Rossum, Monty Python's Flying Circus."
Bron: https://nl.wikipedia.org/wiki/Python_(programmeertaal)

Toepassingsgebieden

Python is geschikt voor een breed scala aan, zo niet alle, toepassingsgebieden. De voornaamste toepassingsgebieden waar Python momenteel wordt gebruikt zijn:

- (BI) Data science
- (ML) Machine Learning
- (AI) Artificial Intelligence
- (IOT) Internet Of Things

Andere toepassingsgebieden zijn o.a.:

- Web applicaties
- Web (REST) API's
- Desktop applicaties
- OS scripting

Kenmerken

In onderstaande tabel worden enkele kenmerken van de Python programmeertaal opgesomd met daarnaast de C# programmeertaal als tegenhanger:

Python	C#
Dynamically typed.	Statically typed.
Duck typed	Strongly typed
Op 'Indentation' gebaseerde code structuur	Op syntax gebaseerde code structuur
Object georiënteerd	Object georiënteerd

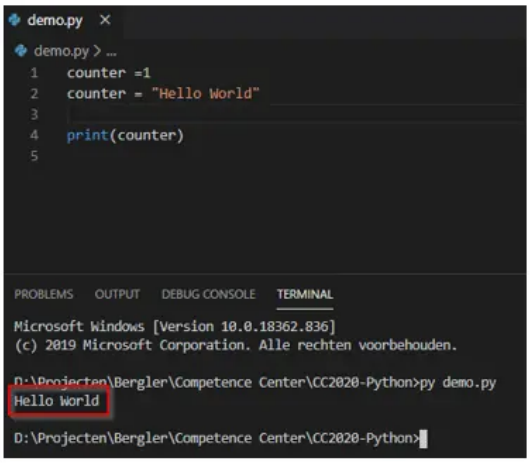
Dynamically typed versus statically typed

In een 'dynamically typed' programmeertaal zoals Python, is het niet noodzakelijk om variabelen met een bepaald type van te voren te declareren. Een variabele kan zelfs tijdens de uitvoering van het programma van type wisselen. In een 'statically typed' programmeertaal zoals bijvoorbeeld C#, dienen de variabelen van te voren, als zijnde van een bepaald type, te worden gedeclareerd.
De onderstaande code is correct in Python, maar zal compiler fouten opleveren in C#:

Python:

```
counter = 1
counter = "Hello World"
print(counter)
```

Deze code zal zonder problemen worden uitgevoerd en print de tekst "Hello world" in de console.



Figuur 1- Dynamically typed voorbeeld

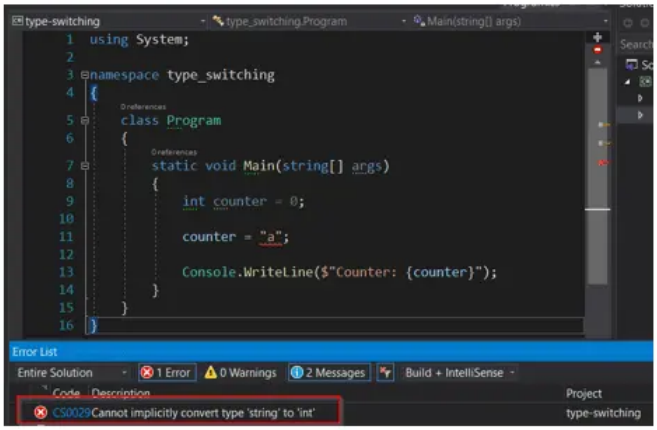
C#:

```
class Program
{
    static void Main(string[] args)
    {
        int counter = 0;

        counter = "a";

        Console.WriteLine($"Counter: {counter}");
    }
}
```

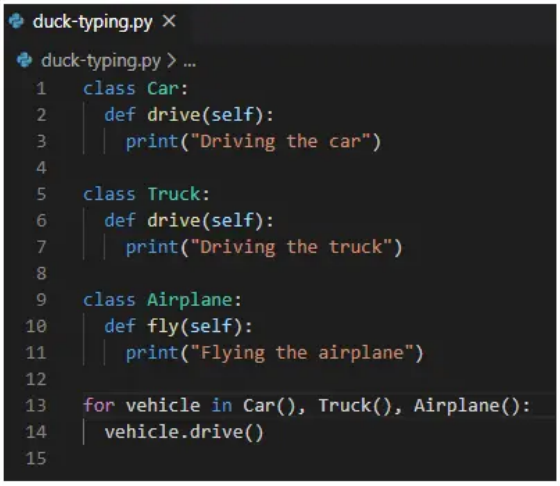
Deze code zal niet compileren en zal dus niet worden uitgevoerd.



Figuur 2- Strongly typed voorbeeld

Duck typed vs Strongly typed

De term 'Duck typing' is voortgekomen uit de zin "If it looks like a duck and quacks like a duck, it is a duck" en wordt voornamelijk toegepast in dynamische programmeertalen. Bij duck typing gaat men ervan uit dat je geen type nodig hebt om een bestaande methode op een object uit te voeren, zolang de methode zelf maar aanwezig is. De volgende python code is een voorbeeld van Duck typing en compileert zonder problemen:



Figuur 3- Duck typing in dynamische taal

Het probleem komt pas boven water tijdens de uitvoering van het programma (run time):

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

D:\Projecten\Bergler\Competence Center\CC2020-Python>py duck-typing.py
Driving the car
Driving the truck
Traceback (most recent call last):
  File "duck-typing.py", line 14, in <module>
    vehicle.drive()
AttributeError: 'Airplane' object has no attribute 'drive'
```

Figuur 4- Runtime error

Omdat C# een strongly typed programmeer taal is zal zulke code direct leiden tot een compiler error, omdat de compiler niet het juiste type van de objecten kan bepalen:

Omdat C# een strongly typed programmeer taal is zal zulke code direct leiden tot een compiler error, omdat de compiler niet het juiste type van de objecten kan bepalen:

```
using System;

namespace Duck_Typing
{
    class Program
    {
        static void Main(string[] args)
        {
            object[] vehicles = new object[] {new Car(), new Truck(), new Airplane()};

            foreach (var vehicle:object in vehicles)
            {
                vehicle.Drive();
            }
        }
    }

    class Car
    {
        public void Drive()
        {
            Console.WriteLine("Driving the car");
        }
    }

    class Truck
    {
        public void Drive()
        {
            Console.WriteLine("Driving the truck");
        }
    }

    class Airplane
    {
        public void Fly()
        {
            Console.WriteLine("Flying the airplane");
        }
    }
}
```

Figuur 5- Compile error

Indentation based vs syntax based

Python is 'Indentation based' de code van een Python programma is gestructureerd d.m.v. 'indentation' (inspringen), terwijl de code van een C# programma gebaseerd is op een vastgelegde syntax. Om binnen Python aan te geven dat een stukje code bij elkaar hoort en gezien dient te worden als 1 code blok gebruik je 'indentation'. De scheiding van de verschillende code blokken wordt aangegeven door 1 of meerdere lege regels.

```python
class Frame():

  def __init__(self):
    self._firstRoll = "-"
    self._secondRoll = "-"

  def setFirstRoll(self, pins):
    if pins > 0:
      self._firstRoll = str(pins)

  def setSecondRoll(self, pins):
    if pins > 0:
      self._secondRoll = str(pins)

  def score(self):
    return str(self._firstRoll) + str(self._secondRoll)
```

Figuur 6- Frame class in Python gebaseerd op 'indentation'

In een op syntax gebaseerde programmeertaal, zoals C# worden de code blokken gedefinieerd doordat de verschillende code regels omsloten zijn door een bepaald symbool, in C# zijn dit de accolades {}.

```csharp
namespace SyntaxBasedExample
{
    public class Frame
    {
        private string _firstRoll;
        private string _secondRoll;

        public Frame()
        {
            this._firstRoll = "-";
            this._secondRoll = "-";
        }

        public void SetFirstRoll(int pins)
        {
            if (pins > 0)
            {
                this._firstRoll = pins.ToString();
            }
        }

        public void SetSecondRoll(int pins)
        {
            if (pins > 0)
            {
                this._secondRoll = pins.ToString();
            }
        }

        public string Score()
        {
            return $"{_firstRoll}{_secondRoll}";
        }
    }
}
```

Figuur 7- Frame class in C# gebaseerd op syntax

Object georiënteerd

Zowel Python als C# zijn programmeer talen waarin je object georiënteerd kunt programmeren. Een van de kenmerken van object georiënteerd programmeren is overerving. Dit kan in Python als wel in C# gerealiseerd worden. De volgende code snippets geven een voorbeeld van een Persoon class en een Student class, waarbij de Student class afgeleid is (overerft) van de Persoon class.

Overerving in Python:

inheritance.py > ...

```python
class Persoon():

    def __init__(self, voornaam, achternaam):
        self._voornaam = voornaam
        self._achternaam = achternaam

    def getFulname(self):
        return self._voornaam + " " + self._achternaam

class Student(Persoon):
    def __init__(self, voornaam, achternaam, afstudeerJaar):
        super(voornaam, achternaam)
        self._afstudeerJaar = afstudeerJaar
```

Figuur 8- Voorbeeld overerving in Python

Overerving in C#:

```csharp
class Persoon
{
    private readonly string _voorNaam;
    private readonly string _achterNaam;

    public Persoon(string voorNaam, string achterNaam)
    {
        _voorNaam = voorNaam;
        _achterNaam = achterNaam;
    }

    public string VolledigeNaam()
    {
        return $"{_voorNaam} {_achterNaam}";
    }
}

class Student : Persoon
{
    private readonly int _afstudeerJaar;

    public Student(string voorNaam, string achterNaam, int afstudeerJaar): base(voorNaam, achterNaam)
    {
        _afstudeerJaar = afstudeerJaar;
    }
}
```

Figuur 9- Voorbeeld overerving in C#

Programmeer-oefening Python

Voor onze interne competence center avond (pizza-avond) hadden we een oefening voorbereid, waarmee onze collega's aan de slag konden om zo wat eerste ervaringen op te doen met programmeren in Python.
Deze oefening is gebaseerd op de wel bekende 'Bowling kata'.
Mocht je interesse zijn gewekt om eens met Python aan de slag te gaan, dan kan je deze oefening hier vinden:
Bowling Kata programmeer oefening.

In het readme.md bestand vind je een beschrijving van de oefening en hoe je hiermee aan de slag kunt gaan.
Veel plezier op je Python ontdekkingsreis!