

# Modul 24

## Objektorientierte Programmierung



VITRUVIUS  
HOCHSCHULE

Oliver Ziegler

# Überblick

- **Namespaces**
- **Unity-Documentation Research**
  
- **Funktionen**
  - **Parameter**
  - **Rückgabewerte**

# Namespaces

## Namespace Vitruvius

```
class Student
```

```
public int semester;
```

```
Student myStudent;
```

Die Klasse Student ist nicht zu finden, weil sie sich im NS Vitruvius befindet

```
Vitruvius.Student myStudent;
```

Explizite Definition zur Nutzung der Bibliothek

Using vereint den Namespace mit der aktuellen Ebene

Ohne Namespace-Angabe nutzbar

```
using Vitruvius;
```

```
Student myStudent;
```

# Namespaces

```
class Student
```

```
public int semester;
```

```
Student myStudent;
```

Bezieht sich auf die selbst definierte Klasse

```
Vitruvius.Student myStudent2;
```

Explizite Definition zur Nutzung der Bibliothek

Namespace Vitruvius

```
class Student
```

```
public int semester;
```

```
using Vitruvius;
```

```
Student myStudent2;
```

Führt zu einer Unklarheit, welche Klasse gemeint ist

# Verschachtelte Namespaces

## Namespace Vitruvius

```
class Student
```

## Namespace GameDesign

```
class Unternehmenspr
```

## Namespace ModeDesign

```
class Kleidung
```

```
Vitruvius.GameDesign.Unternehmenspr uProjekt;
```

## Explizite Variablen - Definition

```
using Vitruvius.GameDesign;  
using Vitruvius.ModeDesign;
```

```
Unternehmenspr uProjekt;  
Kleidung jacke;
```

Using vereint den Namespace mit der aktuellen Ebene

Ohne Namespace-Angabe nutzbar

# Namespaces

## Namespace Vitruvius

```
class Student
```

```
public int semester;
```

```
Student myStudent;
```

Die Klasse Student ist nicht zu finden

```
Vitruvius.Student myStudent;  
ExterneBib.Student myStudent2;
```

Explizite Definition

```
using Vitruvius;
```

```
Student myStudent;  
ExterneBib.Student myStudent2;
```

„using“ des häufig benutzten Namespaces  
Explizit für selten benutzt

## Namespace ExterneBib

```
class Student
```

```
public int klassenStufe;
```

# Documentation Research

- Suchen nach Funktion oder Variable einer Klasse
  - Unity-Dokumentation
  - Internet-Research, Foren
- Beispiel: `UnityEngine.UI.Text``text`

# Google: „Unity Text“

## Unity - Scripting API: Text

<https://docs.unity3d.com/ScriptReference/UI.Text.html> ▾ [Diese Seite übersetzen](#)

`alignByGeometry`, Use the extents of glyph geometry to perform horizontal alignment rather than glyph metrics. `alignment`, The positioning of the text relative to ...

[Text.text](#) · [Text.fontSize](#) · [Graphic](#) · [MaskableGraphic](#)

Du hast diese Seite 4 Mal aufgerufen. Letzter Besuch: 14.04.19

# Google: „Unity Text“

Version: 2018.3 (switch to [2019.1b](#) or [2017.4](#))

- + Navigation
  - Outline
  - PositionAsUV1
  - RawImage
  - RectMask2D
- + Scrollbar
- + ScrollRect
- + Selectable
  - Shadow
- + Slider
  - SpriteState
  - Text**
- + Toggle
  - ToggleGroup
  - VertexHelper
  - VerticalLayoutGroup
- + Interfaces
- + Enumerations
- + UnityEngine.Video
- + UnityEngine.Windows
- + UnityEngine.WSA
- + UnityEngine.XR
- + Classes
- + Interfaces

## Text

class in UnityEngine.UI 7 Inherits from: [UI.MaskableGraphic](#)  
Implements interfaces: [ILayoutElement](#)

Class in **UnityEngine.UI**

### Description

The default [Graphic](#) to draw font data to screen.

### Properties

<a href="#">alignByGeometry</a>	Use the extents of glyph geometry to perform horizontal alignment
<a href="#">alignment</a>	The positioning of the text relative to its RectTransform.
<a href="#">cachedTextGenerator</a>	The cached TextGenerator used when generating visible Text.
<a href="#">cachedTextGeneratorForLayout</a>	The cached TextGenerator used when determine Layout.
<a href="#">flexibleHeight</a>	Called by the layout system.
<a href="#">flexibleWidth</a>	Called by the layout system.
<a href="#">font</a>	The Font used by the text.
<a href="#">fontSize</a>	The size that the Font should render at.
<a href="#">fontStyle</a>	FontStyle used by the text.
<a href="#">horizontalOverflow</a>	Horizontal overflow mode.

# Google: „Unity Text“

Version: 2018.3 (switch to [2019.1b](#) or [2017.4](#))

- + Navigation
  - Outline
  - PositionAsUV1
  - RawImage
  - RectMask2D
- + Scrollbar
- + ScrollRect
- + Selectable
  - Shadow
- + Slider
  - SpriteState
  - Text**
- + Toggle
  - ToggleGroup
  - VertexHelper
  - VerticalLayoutGroup

- + Interfaces
- + Enumerations
- + UnityEngine.Video
- + UnityEngine.Windows
- + UnityEngine.WSA

<a href="#">horizontalOverflow</a>	Horizontal overflow mode.
<a href="#">layoutPriority</a>	Called by the layout system.
<a href="#">lineSpacing</a>	Line spacing, specified as a factor of font line height.
<a href="#">mainTexture</a>	The Texture that comes from the Font.
<a href="#">minHeight</a>	Called by the layout system.
<a href="#">minWidth</a>	Called by the layout system.
<a href="#">pixelsPerUnit</a>	(Read Only) Provides information about how fonts are rendered.
<a href="#">preferredHeight</a>	Called by the layout system.
<a href="#">preferredWidth</a>	Called by the layout system.
<a href="#">resizeTextForBestFit</a>	Should the text be allowed to auto resized.
<a href="#">resizeTextMaxSize</a>	The maximum size the text is allowed to be. 1 = infinity.
<a href="#">resizeTextMinSize</a>	The minimum size the text is allowed to be.
<a href="#">supportRichText</a>	Whether this Text will support rich text.
<b>text</b>	The string value this Text displays.
<a href="#">verticalOverflow</a>	Vertical overflow mode.

**string value the Text displays**

Click



# Google: „Unity Text“

Version: 2018.3 (switch to [2019.1b](#) or [2017.4](#))

- + Navigation
  - Outline
  - PositionAsUV1
  - RawImage
  - RectMask2D
- + Scrollbar
- + ScrollRect
- + Selectable
  - Shadow
- + Slider
  - SpriteState
  - Text**
- + Toggle
  - ToggleGroup
  - VertexHelper
  - VerticalLayoutGroup
- + Interfaces
- + Enumerations
- + UnityEngine.Video
- + UnityEngine.Windows
- + UnityEngine.WSA

```
text.text = „Press space key“;  
Text.fontSize = 48;  
Text.alignment = TextAnchor.MiddleCenter;
```

```
canvas.renderMode = RenderMode.ScreenSpaceOverlay;  
  
// Create the Text GameObject.  
GameObject textGO = new GameObject();  
textGO.transform.parent = canvasGO.transform;  
textGO.AddComponent<Text>();  
  
// Set Text component properties.  
text = textGO.GetComponent<Text>();  
text.font = arial;  
text.text = "Press space key";  
text.fontSize = 48;  
text.alignment = TextAnchor.MiddleCenter;  
  
// Provide Text position and size using RectTransform.  
RectTransform rectTransform;  
rectTransform = text.GetComponent<RectTransform>();  
rectTransform.localPosition = Vector3(0, 0, 0);
```

# Funktionsdefinition

Protection  
keyword

**public**

Rückgabewert  
(void, wenn  
ohne Rückgabe)

**void**

Freiwählbarer Name  
Eindeutig  
Konvention: vorn Groß

**MyFunction**

Parameter  
( ), wenn keine

**( )**



```
{
```

Inhalt/ Befehle, jeweils mit Semikolon

```
}
```

**KEIN SEMIKOLON!**

Die Funktion wird beschrieben, aber nicht ausgeführt, kein Speicher vorbereitet  Kein Befehl

# Return Funktion

```
void MyFunction( )
```

```
{  
  DoStuff();  
  DoSomeMoreStuff();  
  
  return;  
  
  ThisStuffWillNeverBeDone();  
}
```

- Jede Funktion kann durch einen simplen (leeren) return-Befehl beendet werden.

# Rückgabewert

- Ein errechneter oder ausgelesener Wert wird als Ergebnis, wie eine Variable zurückgegeben
- Komplexe Funktionen, die ein Ergebnis bereitstellen, einmal definieren und oft verwenden
- Meist mit Parameter

```
void MyFunction( )
```

```
{  
  string aktuelleZeit = AktuelleZeitAlsText();  
  
  DateTime geburtsdatum;  
  int alter = ErrechneAlter( geburtsdatum );  
}
```

```
string AktuelleZeitAlsText()
```

```
int ErrechneAlter  
  ( DateTime gebDatum )
```

# Funktion mit Rückgabewert

- Definieren dieser Funktionsart
- Datentyp des Rückgabewerts definieren

```
void MyFunction()
```

```
{  
  int eineZahl = 5; GibFuenf( );  
  Debug.Log ( eineZahl );  
}
```

Funktion wird zur Laufzeit durch Rückgabewert ersetzt.



```
int GibFuenf( )
```

```
{  
  return 5;  
}
```

Return-Befehl muss vorhanden sein

# Funktion mit Rückgabewert

- Definieren dieser Funktionsart

void = leer  
(Keine Rückgabe)

**void GreifeGegnerAn()**

```
{  
  schadenGesamt = grundwert + Waffenschaden();  
  
  schadenGesamt = schadenGesamt * SchadenMultiplikatorDurchBuff();  
  
  Gegner.ReceiveDamage( schadenGesamt );  
}
```

Datentyp  
des  
Rückgabe-Wer-  
ts

**int Waffenschaden( )**

```
{  
  int schaden = waffe.schaden + waffe.verzauberung + staerke;  
  return schaden;  
}
```

Rückgabe-  
Befehl

# Funktionsvariablen

## class MyComponent

```
public int myGlobalNumber;
```

### void MyFunction()

```
public int myNumber;
```

```
myGlobalNumber = 10;  
myNumber = 55;
```

### void MyFunction2()

```
public int myNumber;
```

```
myGlobalNumber = 1000;  
myNumber = myNumber + 30;
```

## Klassenvariablen

- Werden auf derselben Ebene wie Funktionen definiert
- Erscheinen im Inspector
- Werden bis zum Szenenwechsel gespeichert

## Funktionsvariablen

- Werden innerhalb von Funktionen definiert
- Erscheinen NICHT im Inspector
- Werden nach der Funktion gelöscht

# Klassen/Funktionsvariablen

## KlassenVariablen

- werden außerhalb von Funktionen definiert
- Sie sind von allen Funktionen ansprechbar (intern und extern)
- Sie werden mit der Instanz gespeichert

## FunktionsVariablen

- werden innerhalb einer Funktion definiert
- Sie existieren nur innerhalb dieser Funktion
- Inhalt/Speicher wird nach dem letzten Befehl wieder gelöscht

# Funktionsvariablen

- Variablen können übrigens schon bei der Definition mit Daten gefüllt werden

```
class MyComponent
```

```
public int myGlobalNumber = 55;
```

```
void MyFunction()
```

```
public int myNumber = 100000;
```

```
public string meinText = „BAM“ ;
```

# Funktionsvariablen

```
class MyComponent
```

```
public int myGlobalNumber;
```

```
void SuperFunction()
```

```
public int myNumber = 5;  
myNumber = 1000;  
myGlobalNumber = myGlobalNumber +  
myNumber;
```

```
void AwesomeFunction()
```

```
public int myNumber;  
  
myGlobalNumber = myGlobalNumber +  
myNumber;
```

Wie groß ist  
„myGlobalNumber“  
am Ende?

```
myGlobalNumber = 20;
```

```
SuperFunction();
```

```
AwesomeFunction();
```

```
SuperFunction();
```

# Funktion: Parameter

- Ermöglichen Funktionen ohne globale Variablen situationsspezifische Aufgaben zu erfüllen

## Funktion 1

```
int x = Stuff();  
MachEtwas2();  
EngineGo();  
Hardware = x;
```

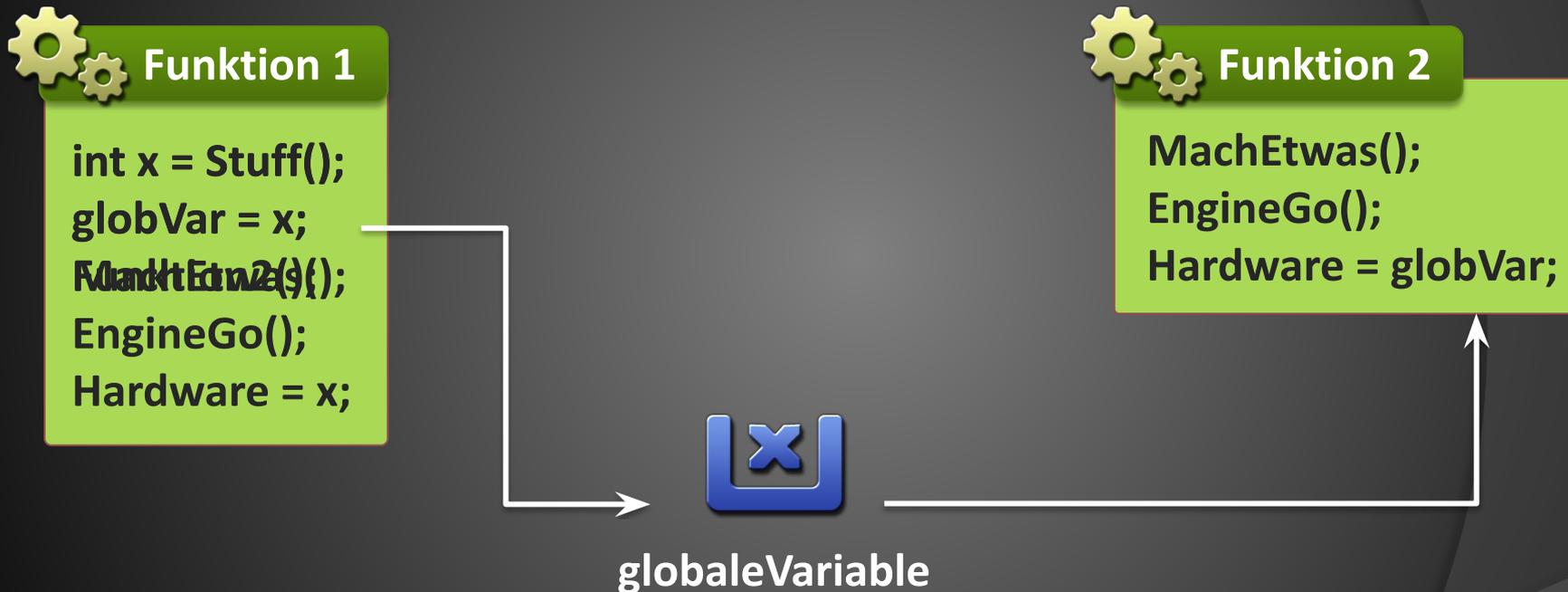
## Funktion 2

```
MachEtwas();  
EngineGo();
```

- Ohne Variablenbenutzung, kann die Funktion 2 nur immer gleiche Befehle ausführen

# Funktion: Parameter

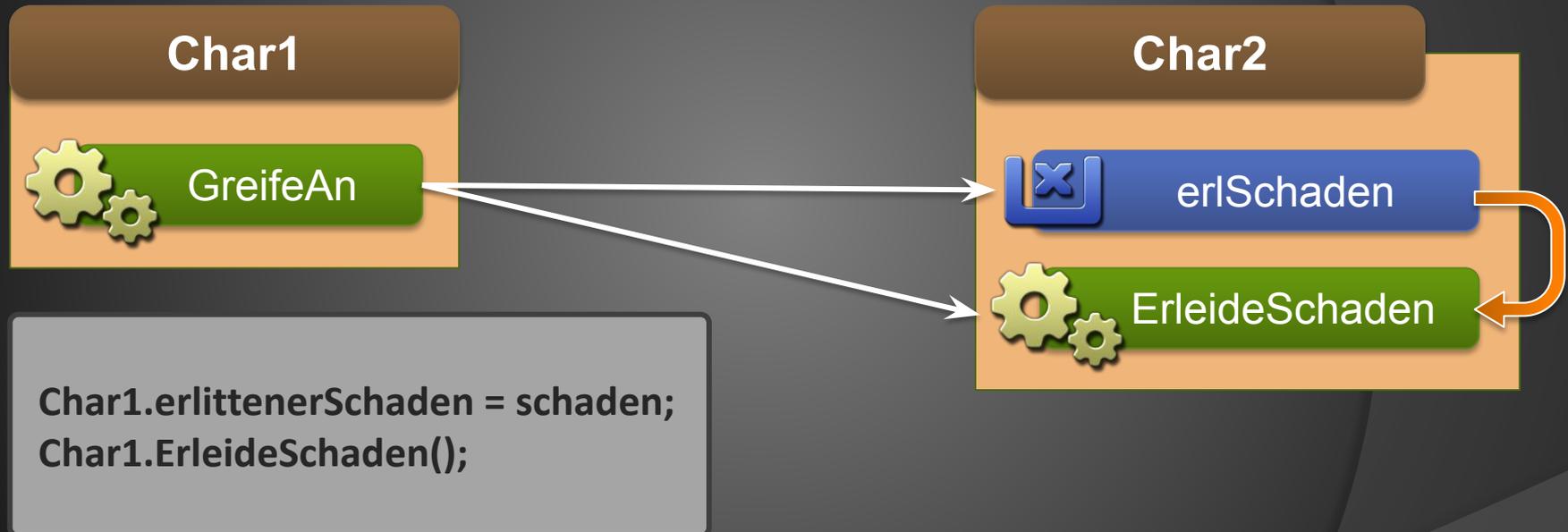
- Ermöglichen Funktionen ohne globale Variablen situationsspezifische Aufgaben zu erfüllen



- Für die Variablennutzung müssten Variablen angelegt werden, die für jeden erreichbar sind

# Funktion: Parameter

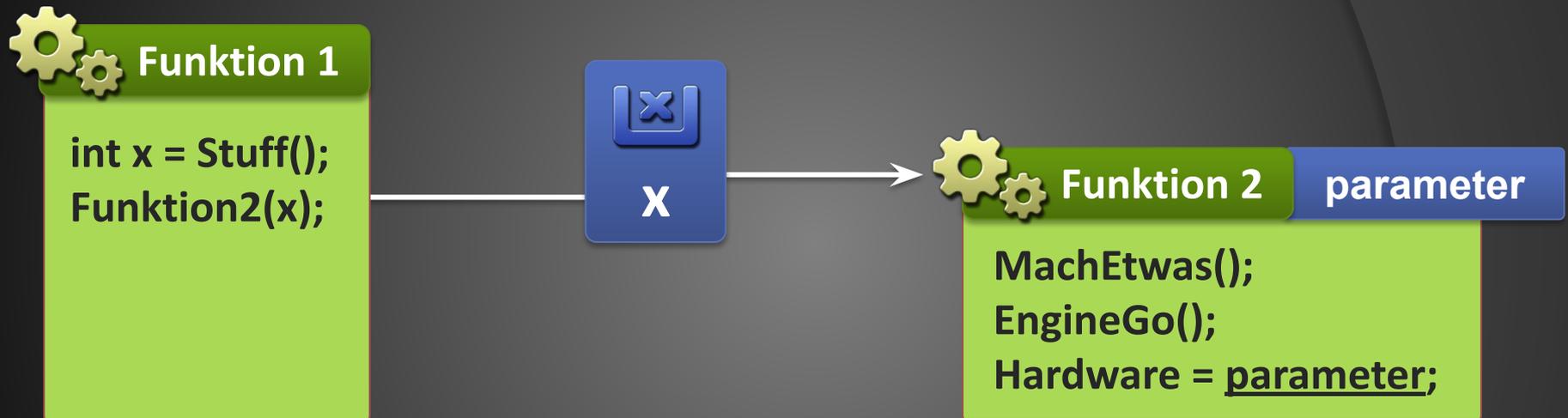
- Ermöglichen Funktionen ohne globale Variablen situationsspezifische Aufgaben zu erfüllen



- In anderen Objekten müssten zunächst Variablen gesetzt und dann die entsprechende Funktion ausgeführt werden
- Diese Variable müsste für jeden zugänglich sein

# Funktion: Parameter

- Ermöglichen Funktionen ohne globale Variablen situationsspezifische Aufgaben zu erfüllen



- Der Wert der Variable wird übergeben, ohne zusätzlichen Befehl
- Wie bei Funktionsvariablen, wird der Parameter nach der Funktion gelöscht.

# Funktion: Parameter

- Erlauben es, Daten als Funktionsvariablen in die Funktion mitzugeben

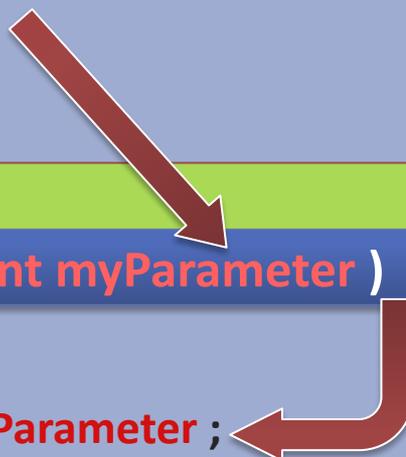
```
public int ergebnis = 55;
```

```
void MyFunction ( )
```

```
{  
    AddiereZuErgebnis( 526 );  
}
```

```
void AddiereZuErgebnis( int myParameter )
```

```
{  
    ergebnis = ergebnis + myParameter ;  
}
```



# Parameter

- Funktionsvariablen, die von außen gesetzt werden

```
Debug.Log( „Das ist ein Test“ );
```

```
public class Debug
{
    public void Log( string text )
    {
        DisplayInConsole( text );
    }
}
```

# Parameter

- Ermöglichen das Auslagern eines Vorgangs in eine andere Funktion
- Erhöhen so Lesbarkeit und senken die Veränderungszeit

```
public void ReceiveDamage( int damage )
{
    text = damage;
    text.animationStyle = 1; // normaler Schaden
    text.StarteAnimation();
    leben = leben - damage;
    if( leben <= 0 )
    {
        leben = 0;
        StarteTodesAnimation();
        ZeigeGameOverScreen();
        LassGegnerLachen();
    }
}
```

# Parameter

- Ermöglichen das Auslagern eines Vorgangs in eine andere Funktion
- Erhöhen so Lesbarkeit und senken die Veränderungszeit

```
public void ReceiveDamage( int damage )
{
    DisplayDamage( damage );
    text.animationStyle = 1; // normaler Schaden
}
text.StartAnimation();
leben = leben - damage;
}
public void DisplayDamage( int damage)
{
    text = damage;
    text.animationStyle = 1; // normaler Schaden
    text.StartAnimation();
}
```

# Parameter

- Ermöglichen das Auslagern eines Vorgangs in eine andere Funktion

```
public void ReceiveDamage( int damage )
{
    DisplayDamage( damage );
    leben = leben - damage;
    if( leben <= 0 )
    {
        leben = 0;
        StarteTodesAnimation();
        ZeigeGameOverScreen();
        LassGegnerLachen();
    }
}
```

# Parameter

- Ermöglichen das Auslagern eines Vorgangs in eine andere Funktion

```
public void ReceiveDamage( int damage )
{
    DisplayDamage( damage );
    leben = leben – damage;
}
```

```
public void AffectDamageOnLive( int damage )
{
    leben = leben – damage;
    if( leben <= 0 )
    {
        leben = 0;
        StarteTodesAnimation();
        ZeigeGameOverScreen();
        LassGegnerLachen();
    }
}
```

# Parameter

- Ermöglichen das Auslagern eines Vorgangs in eine andere Funktion

```
public void ReceiveDamage( int damage )
{
    DisplayDamage( damage );
    AffectDamageOnLive( damage );
}
```

```
public void AffectDamageOnLive( int damage )
{
    leben = leben - damage;
    if( leben <= 0 )
    {
        leben = 0;
        StarteTodesAnimation();
        ZeigeGameOverScreen();
        LassGegnerLachen();
    }
}
```





```
public int leben;  
public int angriff;
```

```
public void GreifeAn( Character targetChar )  
{  
    targetChar.ErleideSchaden( angriff );  
}
```

Objekt-Wechsel

```
public void ErleideSchaden( int schaden )  
{  
    leben = leben - schaden;  
}
```



```
public int leben;  
public int angriff;  
  
public int level;  
  
public void GreifeAn( Character targetChar )  
{  
    int schaden = (angriff * 4) * (level / 2);  
    targetChar.ErleideSchaden( schaden );  
}
```

# Parameter + Rückgabewert

## Charakter



Lebensp.



GreifeAn

## Ausrüstung



Lebensp.



Schlafe

# Parameter + Rückgabewert

```
public void GreifeAn( Charakter gegner )
{
    int schaden = waffe.staerke + waffe.verzauberung + staerke;
    schaden = schaden * level;

    int verteidigung = gegner.vert + gegner.schild.vert;
    verteidigung = verteidigung * gegner.level;

    schaden = schaden - verteidigung;
    gegner.ReceiveDamage( schaden );
}
```

# Parameter + Rückgabewert

```
public void GreifeAn( Charakter gegner )
{
    int schaden = CalculatedDamage( );

    int verteidigung = CalculatedDefense( gegner );

    schaden = schaden – verteidigung;
    gegner.ReceiveDamage( schaden );
}
```

# Parameter + Rückgabewert

```
public void GreifeAn( Charakter gegner )
{
    int schaden = CalculatedMyDamage( );
    schaden = CalculatedDamageAfterDefense( gegner, schaden );
    gegner.ReceiveDamage( schaden );
}
```

```
public int CalculateDamageAfterDefense( Charakter gegner, int schaden )
{
    int verteidigung = gegner.verteidigung + gegner.schild.verteidigung;
    verteidigung = verteidigung * gegner.level;
    schaden = schaden - verteidigung;
    return schaden;
}
```