Logo

Description automatically generatedYOLT! Programming Language



Geschreven door Mathieu te Stroete.

Legenda

[YOLT! 3](#_Toc104978252)

[Introductie van de taal. 3](#_Toc104978253)

[Duidelijke kenmerken van YOLT 3](#_Toc104978254)

[Wat wordt ondersteund door YOLT 3](#_Toc104978255)

[Datatypes 3](#_Toc104978256)

[Operators met priority van laag naar hoog 3](#_Toc104978257)

[Statements en logical operators 4](#_Toc104978258)

[Werken met YOLT 5](#_Toc104978259)

[Hello world 5](#_Toc104978260)

[Variabelen 5](#_Toc104978261)

[Operations 5](#_Toc104978262)

[Short operations 5](#_Toc104978263)

[Operations met verschillende datatypes: 6](#_Toc104978264)

[Statements en Logical operators. 6](#_Toc104978265)

[IF, ELSE IF en ELSE 7](#_Toc104978266)

[Loops 7](#_Toc104978267)

[Methodes 8](#_Toc104978268)

[Input 8](#_Toc104978269)

[Output 8](#_Toc104978270)

[Voorbeeldprogramma’s 9](#_Toc104978271)

[Fibonacci 9](#_Toc104978272)

[FizzBuzz 10](#_Toc104978273)

[Guess The Number! 11](#_Toc104978274)

[Simple class with some functions and statements 12](#_Toc104978275)

[Simple application 14](#_Toc104978276)

# YOLT!

## Introductie van de taal.

De officiële naam van de taal is YOLT! (Yeah, orcs love typing!). In de rest van dit verslag haal ik het uitroepteken aan het eind weg voor leesbaarheid.

YOLT is een taal die het mogelijk maakt voor je huis-tuin en keuken orcs om een simpele applicatie te schrijven. Dat gezegd hebbende, we willen ze natuurlijk niet te veel macht geven aan orcs dus de taal is redelijk primitief. Daarnaast is vrijwel alles in deze taal in hoofdletters, want orcs schreeuwen altijd.

## Duidelijke kenmerken van YOLT

* YOLT is een statische taal. Dit betekent dat je voor al je variabelen het type van tevoren aan moet geven. Daarnaast moet je ook alle variabelen direct een waarde geven in YOLT.
* YOLT ondersteund (globale) variabelen.
* YOLT ondersteund methodes met parameters en return statements.
* YOLT ondersteunt (nested) IF, ELSE IF en else statements.
* YOLT ondersteunt (nested) REPEAT en REPEAT WHILE loops inclusief STOP statement.
* YOLT ondersteund geen classes
* YOLT gebruikt brackets voor scoping.
* YOLT vereist een ; aan het eind van een regel code.
* YOLT ondersteund comments. //\* dit is een comment
* YOLT ondersteund user output met gebruik van SPEAK();
* YOLT ondersteund user input met gebruik van TELL(NUMBER of WORDS);

## Wat wordt ondersteund door YOLT

### Datatypes

|  |  |  |
| --- | --- | --- |
| IDENTIFIER |  | Mogelijke waardes. |
| BOOL | Vergelijkbaar met Boolean in Java | FALSE / TRUE |
| NUMBER | Vergelijkbaar met Integer in Java | elke numerieke waarde. |
| WORDS | Vergelijkbaar met String in Java | Alles wat tussen “ en “ staat. |
| COINS | COINS (Specifiek formaat, gebasseerd op Integer) | 10G\_10S\_10B |

Orcs gebruiken hun eigen munteenheid, daarvoor is in deze taal speciaal COINS aangemaakt. Coins bestaan uit gouden, zilveren en bronzen munten. Een gouden munt is 2898 bronzen munten waard, een zilveren munt is 69 bronzen munten waard. Bij het printen van een munt vertaald de taal dit automatisch naar de maximale hoeveelheid gouden munten. Daarnaast kan op deze manier ook makkelijk vertaald worden hoeveel gouden munten 10000 zilveren munten zijn bijvoorbeeld.

### Operators met priority van laag naar hoog

* + , - (plus, min)
* /, \* (gedeeld door, maal)
* %, ^ (3^3 = 3\*3\*3) (Modulo, tot de macht)
* ()

### Statements en logical operators

|  |  |
| --- | --- |
| Statement operator | Resultaat |
| == | Vergelijkt de waardes. Resulteert in TRUE wanneer deze gelijk zijn. |
| =!= | Vergelijkt de waardes. Resulteert in TRUE wanneer deze niet gelijk zijn. |
| > | Vergelijkt de waardes. Resulteert in TRUE wanneer waarde links groter is dan rechts. |
| < | Vergelijkt de waardes. Resulteert in TRUE wanneer waarde rechts groter is dan links. |
| >= | Vergelijkt de waardes. Resulteert in TRUE wanneer waarde links groter of gelijk is aan rechts. |
| <= | Vergelijkt de waardes. Resulteert in TRUE wanneer waarde links kleiner of gelijk is aan rechts. |
| && | Kijkt of beide statements TRUE zijn, als dit zo is dan resulteert deze statement ook als TRUE. Anders resulteert het in FALSE. |
| || | Kijkt of van statement links en statement rechts een van de 2 TRUE is. Dan resulteert dit statement ook in TRUE. |

# Werken met YOLT

## Hello world

**SPEAK(**"**Hello World**"**);**

## Variabelen

Een globale variabele in YOLT moet altijd boven de eerste functie in de code staan. Hierna staat een voorbeeld van hoe je een normale en globale variabele aan kan maken. Een variabele moet altijd geïnstantieerd worden, dit geld ook voor globale variabelen.

**GLOBAL COINS money = 10G;**

**BOOL value = FALSE;**

## Expressions

**NUMBER value = 8;**

**value = value + 8; (16).**

**value = value – 8; (0).**

**value = value \* 8; (64).**

**value = value / 8; (1).**

**value = value ^ 3; (512), (8\*8\*8)**

**value = value + 8 \* 3 – 5 \* (3+5), (-8)**

## Short Expressions

**NUMBER value = 8;**

**value += 8; (16).**

**value -= 8; (0).**

**value /= value / 8; (1).**

**value \*= 8; (64).**

**value ^= 3; (512), (8\*8\*8)**

## Expressions met verschillende datatypes:

**COINS value = 12G\_10S\_10B;**

**value = value + 8; (12G\_10S\_18B).**

**value = value - 16; (12G\_09S\_63B).**

**STRING tekst = “You have “ + value; (You have 12G\_10S\_10B);**

Gold en Int kunnen met elkaar berekenen, hiervoor is het wel van belang dat het datatype aan de linker kant van de expressie hetzelfde type is als de variabele.

Strings kunnen ook worden opgeteld bij andere datatypes, belangrijk hiervoor is dat de meest linker waarde een string is. De rest van de waardes er dan bij opgeteld. Dit werkt alleen in normale expressies, niet in short expressions.

## Statements en Logical operators.

**IF(Value == 8): (execute codeblock als ze hetzelfde zijn).**

**{}**

**IF(Value =!= 8): (execute codeblock als ze niet hetzelfde zijn).**

**{}**

**IF(Value > 8): (returned true als waarde groter is dan 8)**

**{}**

**IF(Value < 8): (returned true als waarde kleiner is dan 8)**

**{}**

**IF(Value >= 8): (returned true als waarde groter of gelijk is aan 8)**

**{}**

**IF(Value <= 8): (returned true als waarde Kleiner of gelijk is aan 8)**

**{}**

**IF(Value == 8 && value2 =!= 4): (execute codeblock als ze hetzelfde zijn).**

**{} statements klopt).**

**IF(Value == 8 || value2 =!= 4): (execute codeblock als 1 van de**

**{} statements klopt).**

IF, ELSE IF en ELSE  
Een if, else if of else statement mag niet leeg zijn.

**IF(value < 6):{**

**SPEAK(“Value is smaller than 6”);**

**}**

**ELSE IF (value < 8): {**

**SPEAK(“Value is smaller than 8 (and secretly bigger then 6)”);**

**}**

**ELSE: {**

**SPEAK(“Output is bigger then 8.”);**

**}**

## Loops

De taal ondersteund 2 verschillende loops, Repeat en Repeat until. Daarnaast ondersteund het break statements met gebruik van STOP;

**REPEAT(I < 100):**

**{**

**If (I == 20):**

**{**

**STOP;**

**}**

**I += 1;**

**}**

**REPEAT:**

**{**

**If (I == 20):**

**{**

**STOP;**

**}**

**I += 1;**

**}**

**WHILE(condition);**

## Methodes

**FUNCTION Start(Param){ //\* Heeft geen return type. (VOID)**

**}**

**FUNCTION WORDS Start(Param){ //\*vereist return type. WORDS**

**RETURN(“Hello”);**

**}**

**FUNCTION COINS Start(Param){ //\*vereist return type COINS**

**RETURN(5000S);**

**}**

**FUNCTION NUMBER Start(Param){ //\*vereist return type NUMBER**

**RETURN(10);**

**}**

**FUNCTION BOOL Start(Param){ //\*vereist return type BOOL.**

**RETURN(FALSE);**

**}**

## Input

**WORDS message = TALK(WORDS);**

**NUMBERS messageNum = TALK(NUMBERS);**

Output  
Output ondersteund alle datatypes aanwezig in de taal. Daarnaast is het ook mogelijk om + operator uit te voeren in dit statement. Let dan wel op de expressie volgorde en dat je niet iets anders dan een + gebruikt voor het toevoegen van String en een ander datatype.

**SPEAK(“Output”);**

**SPEAK(“Hello user, you have “ + gold\_var + “ of money left!”);**

**SPEAK(“You are awesome: “ + bool\_var);**

# Voorbeeldprogramma’s

## Fibonacci

FUNCTION main()

{

NUMBER num1 = 0;

NUMBER num2 = 1;

NUMBER num3 = 0;

SPEAK(num1);

SPEAK(num2);

REPEAT(num3 < 100):

{

num3 = num1 + num2;

num1 = num2;

num2 = num3;

SPEAK(num3);

}

SPEAK("DONE");

}

## FizzBuzz

FUNCTION main()

{

NUMBER I = 0;

REPEAT(I < 100):

{

IF(I % 5 == 0 && I % 7 == 0):

{

SPEAK("FizzBuzz");

}

ELSE IF (I % 5 == 0):

{

SPEAK("Fizz");

}

ELSE IF (I % 7 == 0):

{

SPEAK("Buzz");

}

ELSE:

{

SPEAK(I);

}

I += 1;

}

}

## Guess The Number!

GLOBAL COINS prizeMoney = 5000S;

FUNCTION main()

{

BOOL found = FALSE;

SPEAK("HELLO, WELCOME TO GUESS THE NUMBER!");

NUMBER amountOfGuesses = 0;

NUMBER toGuess = RANDOM(100);

REPEAT(amountOfGuesses < 7):

{

amountOfGuesses += 1;

SPEAK("IF YOU GUESS IT CORRECT THIS TURN YOU WILL GET " + prizeMoney + "! ");

NUMBER guess = TELL(NUMBER);

IF(guess > toGuess):

{

SPEAK("YOUR GUESS IS TOO HIGH!");

}

ELSE IF(guess < toGuess):

{

SPEAK("YOUR GUESS IS TOO LOW!");

}

ELSE:

{

SPEAK("You guessed the number in " + amountOfGuesses + " tries! Well done! ");

SPEAK("You won " + prizeMoney);

found = TRUE;

STOP;

}

prizeMoney /= 2;

}

IF(found =!= TRUE):

{

SPEAK("You have failed guess the number, better luck next time!");

}

}

## Simple class with some functions and statements

FUNCTION Class() //\* Initializing class for our file which is named class.exLang

{

SPEAK("Hello, this is a simple application that tests some statements and functions.");

}

FUNCTION main()

{

SPEAK("Calculating normalizedPressure. It should result in a value smaller then 11000");

NUMBER normalizedPressure = calculateNormalizedKPA(40, 11000);

SPEAK(normalizedPressure);

SPEAK(" ");

SPEAK("Calculating normalizedPressure. It should result in a value smaller then the previous one");

SPEAK(calculateNormalizedKPA(60, 11000));

SPEAK(" ");

SPEAK("Divide 212 Gold coins and a few bronze and silver coins between 5 people.");

SPEAK(divideCoins(212G\_40S\_11B, 5));

SPEAK(" ");

SPEAK("Divide 5000 Gold coins and a few bronze and silver coins between 42 people.");

SPEAK(divideCoins(5000G\_40S\_11B, 42));

SPEAK(" ");

SPEAK("Divide 212G between only yourself, should result in special message!");

SPEAK(divideCoins(212G\_40S\_11B, 1));

SPEAK(" ");

SPEAK("Divide 50B between 1000 people, should result in special message!");

SPEAK(divideCoins(50B, 1000));

SPEAK(" ");

SPEAK("Crazy statement should result in: Crazy statement is true.");

testCrazyStatements(10, 35, 5);

SPEAK(" ");

SPEAK("Crazy statement should result in: Crazy statement is true.");

testCrazyStatements(20, 24, 10);

SPEAK(" ");

SPEAK("Crazy statement should result in: Crazy statement is true.");

testCrazyStatements(20, 10, 3000);

SPEAK(" ");

SPEAK("Crazy statement should result in num 1, 2 and 3 are smaller then 5!");

testCrazyStatements(1, 3, 4);

SPEAK(" ");

SPEAK("Crazy statement should result in num 1 is smaller then 5!");

testCrazyStatements(1, 30, 4);

SPEAK(" ");

SPEAK("Crazy statement should result in: Doesn't apply to anything.");

testCrazyStatements(30, 10, 50);

SPEAK(" ");

}

FUNCTION NUMBER calculateNormalizedKPA(NUMBER temperature, NUMBER pressureKPA)

{

pressureKPA = pressureKPA \* 293;

temperature += 273;

RETURN(pressureKPA / temperature);

}

FUNCTION COINS divideCoins(COINS total, NUMBER amountOfPeople)

{

IF(total < 100B):

{

SPEAK("Get more money dummy.");

amountOfPeople = 1;

} ELSE IF (amountOfPeople <= 1):

{

SPEAK("Why are you even dividing with only yourself? Kinda weird");

amountOfPeople = 1;

}

RETURN(total/amountOfPeople);

}

FUNCTION testCrazyStatements(NUMBER num1, NUMBER num2, NUMBER num3)

{

IF(num1 > 5 && num1 < 15 || num2 > 15 && num2 < 30 && num2 < 25 || num3 > 1000):

{

SPEAK("Crazy statement is true!");

}ELSE IF (num1 < 5 && num2 < 5 && num3 < 5):

{

SPEAK("Num 1, 2 and 3 are all smaller then 5!");

}ELSE IF (num1 < 5):

{

SPEAK("Num 1 is smaller then 5!");

} ELSE:

{

SPEAK("Doesn't apply to any of the previous statements");

}

}

## Simple application

GLOBAL NUMBER availableSpots = 0;

GLOBAL COINS earnedMoney = 0B;

GLOBAL NUMBER totalAmountOfVisits = 1000;

GLOBAL NUMBER skippedGuests = 0;

FUNCTION Hotel() //\*Initializing class. In this case our file has to be named Hotel.j

{

SPEAK("WELCOME TO YOUR HOTEL SIMULATOR. PLEASE TELL ME HOW MANY ROOMS YOU HAVE");

availableSpots = TELL(NUMBER);

SPEAK("HOW MANY GUESTS DO YOU WANT TO COME?");

totalAmountOfVisits = TELL(NUMBER);

}

FUNCTION main()

{

REPEAT(totalAmountOfVisits > 0):

{

randomBehaviour(2);

randomBehaviour(4);

randomBehaviour(RANDOM(20)); //\* A RANDOM GROUP OF UP TO 20 PERSOONS, who also might just not check in.

NUMBER timeBetweenVisits = RANDOM(10);

REPEAT(timeBetweenVisits =!= 0): //\* Just want to show off that nested loops work.

{

earnedMoney -= 5B;

timeBetweenVisits -= 1;

}

totalAmountOfVisits -= 1;

SPEAK(totalAmountOfVisits);

}

SPEAK("Rooms left in your hotel " + availableSpots);

SPEAK("ALL your guests have visited");

earnedMoney = removeWages(earnedMoney, 2, 2G\_11S\_42B, "Cooks");

earnedMoney = removeWages(earnedMoney, 4, 1500B, "Maids");

SPEAK("Your hotel earned " + earnedMoney + " amount of money!");

IF(earnedMoney > 0B):

{

SPEAK("ME GET TO KEEP " + earnedMoney / 2 + " OF THIS! JAIH!");

}ELSE:

{

SPEAK("WE MADE A LOSS.... IS SECRET! BUT WE LOST " + earnedMoney);

}

}

FUNCTION randomBehaviour(NUMBER groupSize)

{

IF(RANDOM(10) >= 7): //\* check in. RANDOM(BOOL) == TRUE (Alternative version).

{

IF(availableSpots - groupSize >= 0):

{

availableSpots -= groupSize;

earnedMoney = earnedMoney + 2S\_40B \* groupSize;

}

ELSE:

{

skippedGuests += groupSize;

SPEAK("The hotel is full, you are losing money!");

}

} ELSE: //\* check out.

{

availableSpots += 2; //\*This is real bad, and can result in the total amount of spots being bigger then on init.

//\*That said, this is just for showcasing some usecases of the language, not a proper hotel system :).

}

}

FUNCTION COINS removeWages(COINS total, NUMBER employees, COINS pay, WORDS employeeGroup)

{

COINS wages = pay \* employees;

SPEAK("Your " + employeeGroup + " costed you in total " + wages + "!");

RETURN(total - wages);

}