Programmieren 1 - WS 2020/21
Stephan Jokiel
Maxine Rahlfes

Übungsblatt 1

Aufgabe 1

a)

Der **Editor** ist der Bereich, in welchem der Code geschrieben werden kann. Man kann mit hilfe von Operatoren, Objekte bestimmte Dinge tun lassen. (Beispielcodefragment 1_a_1.pf) - Im Beispiel wird “Ich bin ein Beispielcodefragment” ausgegeben.

Die Ausgabe geschieht im “**Out**” bzw. Output. Dort wird ausgegeben, was im Editor bestimmt wurde. Im Beispielcodefragment 1_a_1.pf findet sich der Satz “Ich bin ein Beispielcodefragment” im Output wieder.

Im **Stack** und im **Dictionary** werden z.B. Rechenwege, aber auch Values zwischengespeichert. (Beispielcodefragment 1_a_2.pf) Zugewiesene Werte landen im Dictionary, Rechenergebnisse im Stack.

Das **REPL** (Read, Evaluate, Print, Loop) ist quasi ein kleiner Editor, der einzelne Code-Fragmente instant unmittelbar, ausführt und ausgibt. Die Ergebnisse landen auch wieder auf dem Stack.

Der **Input** dient zur Aufnahme des Codes durch den User. (Beispielcodefragment 1_a_3.pf) Im Beispiel wird der Input aktiviert und fordert den User auf eine line zu schreiben. “Hallo Welt “, dort nun eingegeben, würde in dem Fall “Hallo Welt” lesen, und dies als string im Stack zwischenlegen. Im Beispiel wird nun noch “println” ausgeführt, was das, im Input eingegebene, ausgibt.

Die “**Doc**” ist eine Dokumentation aller Operatoren und funktionsweisen des Postfix IDE.

b)

Die erste Schaltfläche ist mit “**Run**” beschriftet und führt den, im Editor eingegebenen Code aus. Es kann auch “F5” anstelle des Buttons gedrückt werden.

Die **Stepfunktion** (bzw. F6) führt den Code in kleinen Steps aus. D.h. das Programm erkennt einzelne Elemente und führt den Code (wenn man weiter F6 (bzw. den Button) drückt) pro Element (Schritt für Schritt) aus.

Über den Button “**Stop**” kann man während der Ausführung eben diese unterbrechen.

Ein laufendes Programm kann man auch über “**Pause**” pausieren anstelle es ganz Abzubrechen.

c)

Der Menüpunkt **“Add Conditional Breakpoint**” setzt an dem Punkt, an welchem man ihn setzt einen konditionellen Stoppunkt. Dieser pausiert das laufende Programm, bis man es fortführt.

Toggle Breakpoint fügt einen solchen Stopppunkt ohne Kondition hinzu, bzw. entfernt diesen.

Command Palette: Die Command Palette führt eine Liste aller Optionen auf, welche in der Oberfläche des IDE verfügbar sind.
So kann man beispielsweise lines kopieren, verschieben, duplizieren oder aber auch Funktionen zusammenklappen oder wieder auseinanderklappen.

Aufgabe 2:

a) bei der Ausführung „5 4 *“ wird durch die Multiplikation 5 und 4 das Ergebnis 20 im Stack gespeichert
b) bei der Ausführung „4 4 3 +“ steht auf dem Stack 4 und 7. Das lässt sich damit begründen, dass das Programm nur die letzten beiden Zahlen miteinander addiert und dabei die erste 4 bei der Addition auslässt.
c) **Read-int**: dieser Operator kommt zur Anwendung, wenn der Benutzer integer (ganzzahlige) Variablen eingeben soll. Bei nicht integer Variablen reagiert das Programm mit einer Fehlermeldung.

   Read-flt: hier kann der Benutzer auch eine Fließkommazahl in das Inputfeld eingeben.

   Print: mit diesem Operator wird eine Variable/ ein Ergebnis im Output-Feld ausgegeben.

   Println: hiermit wird ebenfalls eine Variable im Output-Feld ausgegeben und gleichzeitig noch ein Zeilenumbruch hinzugefügt.

d) Bei „or“ und „and“ handelt es sich um zwei logische Operatoren im Zusammenhang mit zwei logischen Aussagen, die entweder wahr oder falsch („true“, „false“) sein können. Bei dem Operator „or“ muss mindestens einer der Aussagen wahr sein z.B.: 2 2 + 4 = 3 2 + 6 = or println („true); 2 2 + 5 = 3 2 + 6 = or println (“false”) . Bei dem Operator „and“ müssen beide Aussagen wahr sein, andernfalls handelt es sich um eine falsche Aussage, z.B.: 2 2 + 4 = 7 7 = and println (“true”); 2 2 + 4 = 3 2 + 6 = and println (“false”).(Beispielcodefragment 2_d.pf)
e) (1) 1 2 3 4 + + + = 10

   (2) 1 2 + 3 4 + + = 10

   (3) 1 2 + 3 + 4 + = 10

   (4) 3 4 + 1 2 + + = 10

   (5) 4 3 2 1 + + + = 10

   Alle Aussagen sind äquivalent: (1)=(2)=(3)=(4)=(5). Dies ist möglich, da bei der Addition von Werten die Reihenfolge dieser nicht von Bedeutung ist (Kommutativgesetz, Assoziativgesetz). Bei der Subtraktion wäre dies so nicht mehr der Fall, da hier auf die Reihenfolge der Werte und Operatoren geachtet werden muss, da sonst unterschiedliche Ergebnisse rauskommen.

Beispiel: (1) 1 2 3 4 - - - = -2

(2) 1 2 - 3 4 - - = 0

f) 8 2 + -1 =

= stack: 8

= stack: 8 2

= stack 10

= stack: 10 -1

= stack: false

g) Wenn man nach „clear“ im REPL-Feld eine 1 / eingibt, wird die 1 im Stack gespeichert und im REPL-Feld eine Fehlermeldung angezeigt („Error“).

h) Bei Programm 1: { 2 3 *} werden diese Angeben im stack als Liste gespeichert. Bei Programm 2: [2 3 *] werden diese Angaben als Formel multipliziert und im Stack wird der Wert 6 gespeichert.

i) Das Programm 0.1 0.2 + 0.3 = kommt zum Ergebnis, dass es sich um eine falsche Gleichung handelt. Dieses unerwartete Ergebnis lässt sich damit begründen, dass das Programm auf die unendliche Genauigkeit von Fließkommazahlen achtet und die Zahlen in dem Programm nicht ideal darstellbar sind.

Aufgabe 3: (siehe 3_a.pf bzw. 3_b.pf)

a)
```
read-int
dup
 , -10 >=
 swap
, 0 <=
 , and
print
```

b)
```
 -2 test!
test -10 >= test 0 <= and result!
result println
```