2.

Ausdruck	Wert	Datentyp
100	100	int
(100)	100	int
()	()	tuple
(100,)	(100,)	tuple
(0, 3) + (2, 0)	(0, 3, 2, 0)	tuple
[]	[]	list
2*3*[0,1,1]	[0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1]	list
[1,2,3]+[4,5]	[1, 2, 3, 4, 5]	list
3 in (1,3,3)	True	bool
2/4	0.5	float
2//4	0	int
5|9	13 (0b0101 | 0b1100) == 0b1101	int
9&14	8 0b1001 & 0b1100 == 0b1000	int
3^16	19 0b00011 ^ 0b10000 == 0b10011	int
~7+1	-7 flip bits of 0b00000000000000000000000000000111 (7) binary 0b11111111111111111111111111111000 => -8 -8+1 == -7	int
2<<4	32 0b10 << 4 == 0b100000	int
-9>>2	-3	int
-9<<2	-36 0b11111111111111111111111111110111 << 2 0b11111111111111111111111111011100 => -36	int
1//3+2//3	0	int
1/5+4/5	1.0	float
1.0//3+2.0//3	0.0	float

7**11	1977326743	int
0.1-0.3	-0.19999999999999998	float
0.3+0.1-0.3	0.10000000000000003	float
pow(3,1.5)	5.196152422706632	float
2 % 3	2	int
divmod(2,7)	(0, 2)	tuple
abs(-7)	7	int
complex(5)	(5+0j)	complex
(1+2j)*(3+0j)	(3+6j)	complex
(2+3j)/5j	(0.6-0.4j)	complex
float(abs(4+3j))	5.0	float

3.

Ausdruck	Ergebnis
help()	Öffnet eine Hilfskonsole
import math	Importiert das math-Modul
math.sqrt(2)	Berechnet Square-Root von 2.0 (die 2 wird zu einem float gecastet.
import random	Importiert das random-Modul
random.randint(-100,100)	Random.randint(n0, n1) gibt eine (pseudo) Zufallszahl zwischen n0 (inklusive) und n1 (inklusive) aus.
random.random()	Generiert eine (pseudo) Zufallszahl (float) zwischen 0.0 und 1.0.
usw.	Gibt einen SyntaxError (haha)

4.

Ausdruck	Variablen	Ausgabe	Begründung der Ausgabe
a = [1, 6, 0]	a = [1, 6, 0]		
print(a[1], a[-1])		“6 0”	a[1] ist 6, a[-1] ist gleich a[2] ist gleich 0. **print** fügt zwischen den Werten ein Leerzeichen ein.
b = a	a = [1, 6, 0] b = a		
a[2] = 9	a = [1, 6, 9] b = a		
c = 100	a = [1, 6, 9] b = a c=100		
d = [a, b, c]	a = [1, 6, 9] b = a c=100 d = [a,b,c]		
print(a, d)		“[1, 6, 9] [[1, 6, 9], [1, 6, 9] , 100]”	
a = [b, c, d]	a = [b, c, d] b = unname0 c=100 d=[unname0,b, c] unname0 = [1, 6, 9]		Der Wert von a ändert sich, aber der alte Wert [1, 6, 9] wird weiterhin von b und d referenziert und bleibt somit erhalten. (Ich habe hier den Namen “unname0” gewählt
print(a)		“[[1, 6, 9], 100, [[1, 6, 9], [1, 6, 9], 100]]”	
print(b)		“[1, 6, 9]”	

5.
Kurze, unsichere Version:

```
import math
import sys
sys.argv.pop(0)
a, b, c = map(int, sys.argv)
s = (a+b+c) / 2
print(math.sqrt(s * (s-a) * (s-b) * (s-c)))
```

Längere Version mit vielen Error-checks zur Sicherheit ist als weitere PDF angehängt.

6.
Eine Sprache mit statischer Typisierung fordert, dass die Typen von Variablen und damit von Ausdrücken (Expressions) vor der Laufzeit bekannt sind. Das bringt auch bei Skriptsprachen einen Vorteil, falls neuer Code vor der Ausführung bspw. in einen Zwischencode (Wie JVM-Bytecode oder LLVM) umgewandelt wird.

Es ist wichtig, zwischen explizit statisch typisierten (Wie C) und implizit statisch typisierten Sprachen (Wie Crystal) zu unterscheiden:

Compiler von implizit statisch typisierten Sprachen versuchen, aus den gegebenen Informationen den Typ eines Ausdrucks während vor der Laufzeit zu finden. (Wenn zB. bekannt ist, dass bei “s = string(o)” die Variable s den Typ String hat, kann s im Nachfolgenden Code als String behandelt werden.

Eine weitere Anmerkung: Viele dynamisch typisierte Sprachen (auch Skriptsprachen) versuchen, den Typ von Variablen statisch zu halten, um den Code trotzdem zu optimieren. Ein gutes Beispiel ist Googles’ V8 Engine für JavaScript.

Vorteile:
- Die Fehlerfindung ist für den Programmierer leichter
- Die Fehlerfindung ist für den Compiler leichter
- Der Compiler kann den Code besser optimieren, was zu besserer Performance und weniger Overhead führt (In vielen dynamischen Sprachen haben Variablen ein zusätzliches Feld, über das ihr Typ bestimmt werden kann)

Nachteile:
- Bei vielen statisch typisierten Sprachen ist es umständlich oder unmöglich, den Typ einer Variable zur Laufzeit zu finden
- Der Code ist größer, da der Typ jeder Variable und der Ausgabe jeder Funktion angegeben werden muss
- Besonders bei implizit statisch typisierten Sprachen ist die Zeit länger, die benötigt wird, um den Code zu übersetzen
- Es ist manchmal sehr schwer, ein Programm aus einer dynamisch typisierten Sprache 1-zu-1 in eine statisch typisierte Sprache zu übersetzen