Cheat Sheet:

Chapter 1: Introduction

Algorithm: a step by step list of instructions that if followed exactly will solve the problem under consideration

Python: programming language.

source code: is an instruction in a program that is stored in a file.

high vs low programming lges: high pl needs to be processed before it runs.
vs
low pl if the computer can execute without more processing.

to convert from high programming languages to low: → interpreters
→ compilers

Debugging: tracking down programming errors to correct them.

rrors: 1. syntax: written badly (missing colon, parenthesis...)
2. Run time: when the code crashes.
3. Name: not having defined a variable and then using it
4. semantic errors: it will run, however the answer will be incorrect

ython is a formal language (no ambiguity, less redundant, very literal)

Chapter 2: Python Data:

1. **Data types:** 1. integer: 17 (int)
2. float: 17.5 (float)
3. string: "Hello" (str)

print(type(17)) = integer

2. **Type conversion:** print(int(17.5)) = 17.

3. Variables: Assignment statement: ex: n = 17
m = "Hello"

4. Statements: an instruction, can be specific ex:, if, while, import ...

5. Expression: evaluates to a single value/ outcome.
It can be made of variables, fx, operators.

6. Operators: +, −, *, **, / special ones: → integer division: // (division to nearest integer)
→ modulos operator: % (remainder)

. Order of Operations: BODMAS

$$\begin{array}{cccccc} & (\text{of}) & & (\times) & & (-) \\ & \uparrow & & \uparrow & & \uparrow \\ B & O & D & M & A & S \\ \downarrow & & \downarrow & & \downarrow & \\ (\,) & & (\div) & & (+) & \end{array}$$

Chapter 3: Debugging:

rrors: **parse**: Python couldn't make sense of the structure
ypes **type**: use operation on the wrong type
Name: use undefined variable.
value: type is correct but the value is incorrect.
Syntax: written poorly
import: python can't find or load the module

ips: 1. Use print to know: where your error are from.
2. use the debug icon, and set breakpoints to pause your program + inspect
↳ red dot on the left line
↳ do debug.
3. Comment out code to test: (Ctrl + /) without the code to know if it's an error

Chapter 4: Turtle

Turtle: ex of module in Python

Starts with:

```
import turte (# importing the module)
wn = turtle.screen ()  # creates the window
kampe = turtle.Turtle() # creating a turtled called kampe
  :
wn.exitonclick() # exit on click
```

• couple of movements for the turtle: Kampe.forward ( ) = length
Kampe.right() = clockwise an

2. Herd of turtles: You can create multiple turtles you just need to name them differently

FOR LOOP:

```
→ for x in ["a", "b", "c" ]
      print ("Hi", x , "have a good day")
```

iterations

loop variable

↳ loop body

```
→ for x in range(4 )
      alex. forward(50)
      alex. right (90)
```

drawing a square

* Range function (a , b , x)
x = step.
a = start
b = end.

Chapter 5: Modules

→ ex of modules: turtle, random, screen, math

→ first thing to do is import that module.

→ we can also create our own modules:

→ turtle: do graphics
random: to get random values / outcomes
math: do math operations

Chapter 6: Functions:

A function a sequence of statements

Ex:

```
def square (alex, sz ):
    for i in range( 5):
        alex.forward(sz)
        alex. left (90)
```

→ function

→ inputs.

Assert: perform a unit test of True or False

Nested loops: A loop inside a loop

Direction of flow: 1. The outer loop runs first
2. For each iteration of the outer loop the inner loop runs completely.

```
ex: for outer in range(2):
        for inner in range (3):
            print(outer, inner)
```

→ i

↳ j

so outer loop i = 0 → inner loop runs j = 0, 1, 2
(0,0) → (0,1) → (0,2)
outer loop i = 1 → inner loop runs j = 0, 1, 2.
(1,0) → (1,1) → (1,2)

total iterations $= i \times j = 2 \times 3 = 6$

⚠ Don't forget indentation. for outer loop and another indentation for inner loop