The Python Cheatsheet

Printing

- To print, we use the print command
- The command works as follows:

```
1  name = "Emma"
2  print("hello world")
3  print("I am:", name)
4  print("I am: " + name)
5  print(f"I am: {name}")
6  print("I am: {0}".format(name))
```

which returns →

```
hello world
I am: Emma
I am: Emma
I am: Emma
I am: Emma
```

- Lines 3, 4, 5 and 6 all show the same thing as output, but are all written differently
- Lines 5 and 6 are the 'best' methods, I will explain how line 5 works, 6 is quite similar:

```
5  print(f"I am: {name}")
```

This is what we call an f-string. It is a special way to add a variable into a string in the most concise manner possible.

We use these curly brackets '{' and '}' to enclose the variable we want to print

- We can use this style to print as many things as we like, for example:

```
name = "Emma"
age = 99
noSiblings = 1
print(f"I am {name} and I am {age} year(s) old and I
    have {noSiblings} sibling(s)")
```

→

```
I am Emma and I am 99 year(s) old and I have 1 sibling(s)
```

If-statements

- If statements allow us to alter the flow of the program (what gets executed) depending on specific conditions that we desire. For example:

```
age = 12

if age >= 18:
    print("you are an adult")
else:
    print("you are not yet an adult")
```

→ `you are not yet an adult`

- We set age equal to 12, and as our if statement only wants values where age ≥ 18, the code beneath the else is executed.

- Now, lets look at an enhanced version of the if statement, using elif. Elif (else if) allows us to test for multiple conditions, not just one as in the example above. For example:

```
1  age = 15
2
3  if age >= 18:
4      print("You are an adult")
5  elif age >= 13:
6      print("You are a teenager")
7  else:
8      print("You are a child")
```

→ `You are a teenager`

- By using the elif, we were able to test for those in the age ranges that are >=13 but <18. You should also note that once an if statement is matched, the remainder of the elif/else statements will not be checked - which is why the ordering of the statements matter!
- If you'd like to, you can write as many elif statements as you like (though if there are a lot of them, there are probably better ways of doing what you're trying to do).

FOR-loops

- For loops allow you to repeatedly execute a set of instructions for a known/discoverable amount of times. They can be used to iterate (go over) over just about everything from data structures (more of that later) to integers to strings. Lets see some examples:

Example 1: Using integers

- There are thousands of different ways to use integers, this is only one:

```
for i in range(10):
    print(i)
```

→

```
0
1
2
3
4
5
6
7
8
9
```

Python will execute this 10 times, but Python ALWAYS starts from 0, so as you can see on the right, you are seeing the values 0 to 9.

How does this work behind the scenes?

- Python does a lot of work behind the scenes for you, and a for-loop is one example of that.

```
for i in range(10):
    print(i)
```

i=0

i+=1

delete i

- When the for loop is first entered, Python creates a variable i, assigned the value 0 (if we changed how the range function worked this could be different). At the end of every iteration, or in this case after each time we print i, Python secretly increments i (adds 1) to it until it reaches the maximum value. At this point, the for loop is complete and i is deleted.

Example 2: Using Strings

- Lets say you wanted to count every vowel in a string, we can do that using a for loop.

```
1  word = "Emma <3 Comp Sci"
2  noVowels = 0
3  other = 0
4
5  for letter in word.lower():
6      if letter in "aeiou":
7          noVowels+=1
8      else:
9          other += 1
10 print("This is the number of vowels: {0} and this is
       the number of others: {1}".format(noVowels,
       other))
```

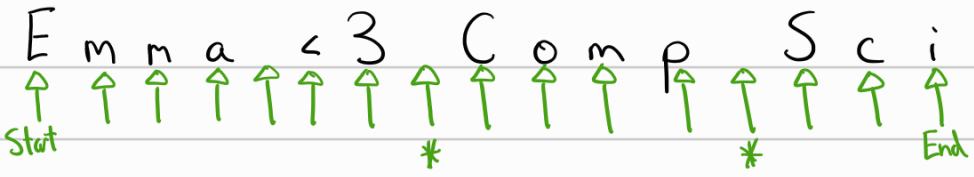
this is a special function to make a string lowercase

```
This is the number of vowels: 4 and this is the number of
    others: 12
```

this can be named anything you like, but the more logical your naming is, the easier it is to read!

How does this work behind the scenes?

- A string is just a list of individual characters (we will see lists later on) but what is essetially happening is like this:



- With every loop, the ficticious green arrow 'jumps' to the next character, and stores that in the variable we called letter. Remember that whitespace (the empty space that seperates words, shown with *) is also a character, so make sure you consider that when writing code!

WHILE Loops

While loops are very similar to for-loops, but they give us more freedom to create a wider variety of loops. For loops are typically counter-controlled loops, whilst while loops are condition-controlled (a specific event needs to occur for the loop to stop). We can replicate for-loops as while loops, but not necessarily the other way round.

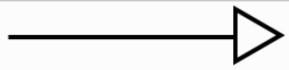
Example 1: Replicating a for-loop as a while-loop

Lets start by using this simple for loop:

```
for i in range(10):
    print(i)
```

In order to turn this into a while loop, we need to understand and replicate what Python does behind the scenes for us. That is, we need to now explicitly create the variable i, and increment it, giving us the following:

```
1  i = 0
2  while i != 10:
3      print(i)
4      i += 1
```



```
0
1
2
3
4
5
6
7
8
9
```

This is an example of a condition. That is, the code will run until a specific event has occurred, which in this case, the value of i is printed until it is equal to 10, at which point we stop. A nice way to think about a while loop is that you are doing something until something else tells you to stop, whilst a for-loop is for when you know how many times you want to do something for. A real life example is:

while the kitchen is dirty:
    clean()

↑ we don't know how long this will be, perfect for a while loop!

for every apple at home:
    eat()

↑ We know that the number of apples is countable, perfect for a for loop

Lets try replicating the more difficult example now:

```
word = "Emma <3 Comp Sci"
noVowels = 0
other = 0

for letter in word.lower():
    if letter in "aeiou":
        noVowels+=1
    else:
        other += 1
print("This is the number of vowels: {0} and this is the number of others: {1}".format(noVowels, other))
```

=

```
words = "Emma <3 Comp Sci"

counter = 0
stringLength = len(words)

noVowels = 0
other = 0

while counter != stringLength:

    if words[counter].lower() in "aeiou":
        noVowels+=1
    else:
        other += 1

    counter += 1
print("This is the number of vowels: {0} and this is the number of others: {1}".format(noVowels, other))
```

As you can see, this change to a while loop has resulted in us needing to write much more code simply to do the same thing as before - which is why it is important to choose the nicer and more easy to understand way.

P.S: At this point in time, you probably will not understand fully what the code in the while loop is doing, and that is okay! We will better understand this later on.

Example 2: A different type of while statement

- Lets say we wanted to get the user (the individual using your program) to only be allowed through if they write the correct password. As we don't know how many attempts this may take, it is perfect for a while loop!

```
myPassword = "ilovecomputerscience"

passwordEntered = input("Please enter the correct password: ")

while passwordEntered != myPassword:
    print("That is the incorrect password. Please try again.")
    passwordEntered = input("Please enter the correct password.: ")

print("Success! System Access Granted!")
```

→

```
Please enter the correct password: iloveenglish
That is the incorrect password. Please try again.
Please enter the correct password.: ilovemaths
That is the incorrect password. Please try again.
Please enter the correct password.: ilovecomputerscience
Success! System Access Granted!
```

As you can see, once we put in the correct password, we were granted access to the system! You may wonder, why are lines 3 and 7 exactly the same? This is because Line 3 is only executed once and its content (the inputted password) is used to start the while loop, whilst line 7 asks the same question again using the same variable name so that we can keep asking over and over again until it is correct, updating passwordEntered correctly.

Your Exercises

1. Fizzbuzz

Given an inputted number by the user, I would like you to print the following:

- if the number is divisible by 3, "Fizz"
- if the number is divisible by 5, "Buzz"
- if the number is divisible by 3 and 5, "FizzBuzz"
- otherwise, "not divisible by 3 or 5"

2. Guessing game

I would like you to choose a number and I would like you to ask the user for a number. I want you to keep track of the number of times the user inputs the incorrect number, and once the correct number is guessed, to return how many guesses it took.

Need some help? Hints are on the next page!

Exc 1: FizzBuzz

Hint 1: maybe the % (modulo) operator will be useful here. Remember that the modulo operator returns the remainder of the division of two numbers. E.g:

$5 \% 3 = 2$

$4 \% 2 = 0$

$20 \% 15 = 5$

```
age = 15
if age <= 18 and age >= 13:
    print("You are a teenager")
```

→ You are a teenager

```
number = 8
if number % 2 == 0 and number < 10:
    print("The number is even (divisible by 2) and less than 10")
```

→ The number is even (divisible by 2) and less than 10

Hint 2: It seems like one of our if statements will need to use the 'and' keyword to check for both 3 and 5. The following are examples of how and works:

Exc 2: Guessing game

Hint 1: It seems like we do not know how may guesses it may take until the user gets it right. Perhaps a while loop is best here, and the first example in Example 1 and Example 2 in the While-loop section will help!

Hint 2: We are going to need to keep a counter variable that is incremented within the while loop, as well as a way for the user to try again until they get it correct. It may be useful to write out all the steps before writing actual code.