Session 3: For loops and while loops

The need for looping

Imagine you have to print a square of *s using Python.

```
**********
*         *
*         *
*         *
*         *
*         *
*         *
*         *
*         *
**********
```

One solution is to do this:

```
print("**********")
print("*         *")
print("*         *")
print("*         *")
print("*         *")
print("*         *")
print("*         *")
print("*         *")
print("*         *")
print("**********")
```

But don't you think it is too repetitive? This is when loops come in handy. They allow you to repeatedly run the same piece of code multiple times. There are two types of loops, for and while loo;s.

For loops

Runs the body a specified number of times.

For example, to print the same square as the one above, we can do:

```
print("**********")
for i in range(8):
    print("*         *")
print("**********")
```

`range(8)` means that the statements indented will be run 8 times, in other words, 8 **iterations**. Indentation is important to specify which statements are in the for loop and which ones are not.

What is i?

Let's try to print the value i in every iteration.

```
for i in range(5):
    print(i)

# 0
# 1
# 2
# 3
# 4
```

The variable `i` consecutively takes all values from 0 to n-1, it helps us to keep track of our progress while looping, the following example uses the value `i` to print out the first `n` non-negative even numbers.

```
n = int(input("Input a number: "))

for i in range(n):
    print(i*2)
```

It is common to use variable names like `i`, `j`, `k` for iteration.

It is also common for us to be wanting to iterate from 1 to n instead of from 0 to n-1. In this case, we can use `range(1,n+1)`, meaning that it will loop from 1 to n+1, including 1 but excluding n+1. That means looping from 1 to n inclusive. The following code prints out the first `n` positive even numbers. (excluding 0)

```
n = int(input("Input a number: "))

for i in range(1,n+1):
    print(i*2)
```

Short aside: incrementing numbers

We've seen before that we can do:

```
number = 1
print(number + 1) # Prints '2'
print(number) # Prints '1'
```

But what if we want to actually increase the values of `number`? We might be tempted to do:

```
number = 1
number + 1
print(number) # Still prints '1'
```

But that doesn't change `number` at all: it just asks Python to evaluate what `number + 1` is. The way to actually increment `number` is:

```
number = 1
number = number + 1
print(number) # Prints '2'
```

This might seem a bit confusing at first, since `number` appears both on the left and on the right of the `=`. The rule in Python is that stuff on the right side of and `=` are evaluated before stuff on the left side. You can think about the above code like it was shorthand for this:

```
number = 1
temp = number + 1
number = temp
print(number)
```

Another way to do this exact same thing is:

```
number = 1
number += 1
print(number)
```

Here, the `+=` operator is used as a shorthand for writing `number = number + 1`. These kinds of shorthands are usually referred to as *syntactic sugar* in programming - it's shorter so it's "sweeter" for people to read and write. There are analogous operators: `-=`, `*=`, `/=`, etc.

```
number = 1
number += 2
number -= 3
number *= 4
number /= 5
# ...
```

If unsure, just use the first form: `number = number + 1`.

In the specific case when we have `number += 1` (i.e. we have `1` on the right-hand side) we say that we **increment** `number`. When we do `number -= 1` we say that we **decrement** `number`.

Worked example: summing from 1 to `n`

The following example sums the first `n` natural numbers, including `n`.

```
n = int(input("Please enter a number: "))

sum = 0
for i in range(1,n+1):
    sum += i

print("The sum of the first " + str(n) + " natural numbers is " + str(sum))
```

Exercises

Easy

1. What are the outputs of the following program segments?

```
# a)
for i in range(5):
    print(i)
```

```
# b)
for i in range(4,10):
    print(i)
```

```
# c)
for i in range(5):
    print("Hello")
```

```
# d)
for i in range(4,10):
    print("Hello")
```

When attempting these exercises, I suggest you type out the code by hand, rather than copy-pasting it. Pay attention to all of the symbols you are typing and see if you can recall why they are there.

2. Complete the program below, which outputs the first `n` powers of 2.

```
n = ___(input("Input a number: "))
number = 1
```

```
for i in range(_____):
    number_____________
    print(number)
```

```
Sample:
Input a number: 5
2
4
8
16
32
```

3. Write a similar program, which outputs the `n`th power of 2. (should work for all n >= 0)

```
Sample:
Input a number: 5
2 to the power 5 is 32.
```

Medium

4. Write a program, which asks the user for an input `n` and then for `n` numbers. Then, it prints out the average of those `n` numbers. A run of your program might look something like this:

```
How many numbers? 3
Please enter a number: 1
Please enter a number: 3
Please enter a number: 5
The average was: 3.0
```

5. Complete the program below, which outputs the first `n` fibonacci numbers. (Each number is the sum of the two preceding ones, starting with 0,1.)

```
t = ______(input("Input a number: "))
a = 0
b = 1
for ______ in range(t):
    c = a + b # calculate new term
    #shift the terms by 1
    a = ____
    b = ____
print("Term " + str(i) + " is " + _____________)
```

```
Sample:
Input a number: 5
Term 0 is 0
Term 1 is 1
Term 2 is 1
Term 3 is 2
Term 4 is 3
Term 5 is 5
...
Term 14 is 377 (just for checking)
```

Hard

6. Create a program, which takes as input a number `n` and then prints the following:

   1

   1 2

   1 2 3

   .....

   1 2 3 4 ... n

   Hint: If you want to print something without a newline at the end, the way to to that is:

```
number = 10
print(number, end=" ") # This will print " " after number.
```

7. a) Create a program, which prints a square of *s of size `n` based on input. As a generalisation of the example.

```
Input side length of the square: 10
**********
*        *
*        *
*        *
*        *
*        *
*        *
*        *
*        *
**********
```

   b) Try not to use string "multiplication" and string "addition".

   c) Try modify your program so that it only outputs 1 star when n is 1, and no stars when n <= 0

You can find all the solutions here.

While loops

Let's look at another way to print the square:

```
print("**********")
i = 0
while i<8:
    print("*         *")
    i+=1
print("**********")
```

The idea with the while loop is very simple: the code in the body of the loop will be executed repeatedly, while the condition remains true; otherwise said: until the condition becomes false. In each iteration, i got incremented by 1, until it reaches 8 and the loop quits. Hence the loop is run 8 times.

We can use while loops to do more things. Imagine we wanted to create a simple security program, which asks our user for a password, until they enter the correct one. One way to do this is:

```
SECRET_PASSWORD = "pass123"

user_input = input("Please enter a password: ")

while user_input != SECRET_PASSWORD:
  print("Access denied: wrong password.")
  print() # Print a newline. This is just here to make things look nicer.
  user_input = input("Please enter the password: ")

print("Access granted.")
```

In this case, while the user keeps entering the wrong password, the program will keep prompting them for a new one.

Now, if the user enters the correct password on their first try, the body of the loop will not be executed.

Worked examples

Let's see some more examples. Here is how we print the first `n` positive integers:

```
n = int(input("Please enter a number: "))

i = 1
while i <= n:
  print(i)
  i += 1 # Increment i (i becomes bigger by 1).
```

Again, the variable `i` consecutively takes all values from 0 to n-1, it helps us to keep track of our progress while looping, the following example uses the value `i` to print out the first `n` non-negative even numbers. It is common to use variable names like `i`, `j`, `k` for iteration.

Here is how we calculate the sum of the first n natural numbers:

```
n = int(input("Please enter a number: "))

i = 1
sum = 0
while i <= n:
  sum += i
  i += 1

print("The sum of the first " + str(n) + " natural numbers is " + str(sum))
```

A common pitfall is to forget to increment. Here is how we **don't** calculate the sum of the first n natural numbers:

```
n = int(input("Please enter a number: "))

i = 1
sum = 0
while i <= n:
  sum += i

print("The sum of the first " + str(n) + " natural numbers is " + str(sum))
```

If that looks like the same code to you, don't worry; even experienced programmers often make this simple mistake. The issue is that we don't increment `i` within the body of the loop. So every time we pass through the loop, `i` remains `1`. If `n` is more than `2`, then this loop will go on forever. **This is a very common mistake. Always keep this in mind if your program seems to "hang".**

For loops and while loops

Yes we can do everything we can do with for loops using while loops. But for loops are useful because `for` loops take care of a lot of things for us - you don't have to define your own `i` and increment it yourself. On the other hand, `for` loops less flexible - there are some cases where you might not want to have a counter or you might want to decrement it, hence there are some cases that only `while` loops can be used.

It is up to you which kind of loop you are going to use based on the situation, sometimes you can only use `while` loops.

Exercise: Implement some of the exercises you have done in the for loops section using while loops.

Exercises:

Easy:

8. What are the outputs of the following program segments?

```
# a)
i = 5
while(i>0):
    print(i)
    i-=1
```

```
# b)
i = 5
while(i>=0):
    print(i)
    i-=1
```

```
# c)
i = 5
while(i>0):
    i-=1
    print(i)
```

```
# d)
i = 5
while(i>=0):
    i-=1
    print(i)
```

```
# e)
i = 10
while(i>0):
    print(i)
    i-=3
```

```
# f)
i = 5
while(i>0):
    i-=1
    print("Hello")
```

```
# g)
i = 10
while(i>=0):
    print(i)
    i-=2
```

```
# h)
i = 0
while(i<20):
    i+=4
    print(i)
```

When attempting these exercises, I suggest you type out the code by hand, rather than copy-pasting it. Pay attention to all of the symbols you are typing and see if you can recall why they are there.

9. Create a program that asks the user to input a number `n`, and then prints the first `n` positive integers in reverse order. E.g. if `n` is 4, your program should print: '4 3 2 1', each number being on a new line. (*Hint: model your program after one of the programs we already wrote.*)

```
Sample:
Please enter a number: 4
4
3
2
1
```

10. Complete the program below, which asks the user to input a number `n` and then prints the product of the first `n` natural numbers.

````
```python
n = int(input("Please enter a non-negative integer: "))

product = 1
i = 1
while # Complete the rest.
    # ...

print(product)
```

```
Sample:
Please enter a non-negative integer: 4
24
```
````

11. Complete the program below, which asks the user to input a number `max` and then prints all square numbers until `max`.

````
```python
n = int(input("Please enter a number: "))

base = 1
while # Complete the rest.
    print #...
    base #...
```

```
Sample:
Please enter a number: 100
1
4
9
16
25
36
49
64
81
100
```
````

Medium:

12. Create a program, which repeatedly asks the user to input a number, until their number is greater than 9. A run of your program might look like:

````
```
Please enter a number: 5
That number is too small!
Please enter a number: 9
That number is too small!
Please enter a number: 10
That's a good number!
```
````

13. Create a program, which continuously prompts the user to input a first name and a last name, until the names they enter match yours.

14. Modify the code you wrote for question 10, give an error message if user inputs a negative number, and asks the user to input again.

```
Sample:
Please enter a non-negative integer: -99
Invalid number! Please enter again.
Please enter a non-negative integer: -3
Invalid number! Please enter again.
Please enter a non-negative integer: 4
24
```

Hard:

15. (Math) Write a program, which calculates the average of numbers. The program terminates when -1 is inputted.

```
Please enter a number: 1
Please enter a number: 3
Please enter a number: 5
Please enter a number: -1
The average was: 3.0
```

You can find all the solutions here.