Session 3: While loops and Try

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.

So, while loops tend to be more useful if the body of the loop has to loop an unknown number of times.

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:

1. 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.

2. 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
```

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

```
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
```

4. 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:

5. 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!
```

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

7. Modify the code you wrote for question 3, 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:

8. (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.

Try

(New section added this year)

In the password checking loop at the top of this section, we saw how a program could protect itself from incorrect input, to some extent. But the program still breaks if the user does something wildly incorrect such as

```
favorite_number = int(input("Please enter your favorite number."))
# User enters the word "Puppy"
# Program crashes!
# ValueError: invalid literal for int() with base 10: 'Puppy'
```

The try keyword is a stronger way of protecting a program against incorrect input. Programmers tend to worry a lot about making sure nonsensical user input can't mess up their precious programs (but for all that worrying, they get it wrong anyway)

For example, here's some code that makes sure that checks if the user inputted an integer.

```
user_input = input("Please input an integer.")

try:
    user_input = int(user_input)
    print("That was an integer.")
```

```
except:
    print("That was not an integer.")
```

Inside the `try` block, Python *tries* to convert the user's input into an integer. If it suceeds, the rest of the code in the `try` block can run, which means Python prints "That was an integer." But if the conversion fails, Python panics.

Usually, when Python panics, you get an error message on your screen. This is called **throwing an exception**. But because the error happened inside a `try` block, Python understands that this error was accounted for, and calmly moves on to the `except` block, where it prints "That was not an integer."

Worked Example

Here's how we can use `try` to wait until the user inputs an integer.

We start with the code from before.

```
user_input = input("Please input an integer.")

try:
    user_input = int(user_input)
    print("That was an integer.")
except:
    print("That was not an integer.")
```

Now, in the except block, we want to ask the user for another input, and keep asking them until they get it right. In other words:

```
user_input = input("Please input an integer.")

try:
    user_input = int(user_input)
    print("That was an integer.")
except:
    # while... the user's input is wrong
    #   input... ask them for an input
    #   try... converting the input
    #   except...
```

It's hard to tell how to check that "the user's input is wrong" here. We could set a variable inside the `except` block:

```
user_input = input("Please input an integer.")

try:
    user_input = int(user_input)
    print("That was an integer.")
except:
    user_input_is_wrong = True
    while (user_input_is_wrong):
        user_input_is_wrong = False
        # input... ask them for an input
        try
            # convert the input
            user_input_is_wrong = False
        except
            user_input_is_wrong = True
```

But then we run into the question of "How do we get the converted input value out of the try/except block?"

The answer is to create a new variable outside the try/except block that we can write to at any time:

```
user_integer = None
user_input = input("Please input an integer.")

try:
    user_integer = int(user_input) # This line also changed!
except:
    user_input_is_wrong = True
    while (user_input_is_wrong):
        user_input_is_wrong = False
        user_input = input("That was not an integer. Please try again.")
        try
            user_integer = int(user_input)
            user_input_is_wrong = False
        except
            user_input_is_wrong = True
```

Actually, if the `user_integer` is definitely not going to be `None` if it gets correctly converted, we can use `while (user_integer == None)` in our while loop, which removes the need for the `user_input_is_wrong` variable as well!

```
user_integer = None
user_input = input("Please input an integer.")

try:
    user_integer = int(user_input) # This line also changed!
except:
    while (user_integer == None):
        user_input = input("That was not an integer. Please try again.")
        try
            user_integer = int(user_input)
        except
            pass # Python complains if this block is empty, so we use the
"pass" keyword, which means "do nothing".
```

Actually, we can remove the outer `try`/`except` block as well !!!

```
user_integer = None
user_input = input("Please input an integer.")

while (user_integer == None)
    try:
        user_integer = int(user_input)
    except:
        user_input = input("That was not an integer. Please try again.")
```

(This exchanged a redundant line of code for a redundant check when the code runs, since `user_integer` is definitely going to be `None` when we first enter the `while` loop.)

Exercises

Easy

1. (Open Ended) Find out what the `else` keyword does when used alongside a `try`/`except` block. (Put it at the end!) Use this information to enhance the "enforce integer input" code above in any way you like.

2. Add a counter to the "enforce integer input" code to show the user how many times their input has been incorrect.

Medium

3. (Open Ended) Find out what the `finally` keyword does when used alongside a `try`/`except` block. Try to think of a situation where the `finally` keyword might be useful.

4. Create a program that accepts user inputs until the user has inputted two integers.

   ```
   Input: 4
   Input: score
   Input: and
   Input: 7
   You have inputted the maximum allowed number of integers. Goodbye!
   ```

5. Create a "pick from a list" block of code that only accepts an input from 1 to some number.

   ```
   Please choose an option: 9
   That is not an option. Please try again: wow
   That is not an option. Please try again: 2
   Option 2 selected!
   ```