

Week 2: Programming Fundamentals Continued

If statements

An if statement only executes the code in its **body** if the **condition** is true.

```
a = int(input())

if a > 5:
    print("This number is big.")
    print("This only gets printed if a is larger than 5.")

print("This always gets printed.")
```

Green is input. White is output.

8

```
This number is big.
This only gets printed if a is larger than 5.
This always gets printed.
```

3

```
This always gets printed.
```

If statements

An if statement only executes the code in its **body** if the **condition** is true.

```
a = int(input())  
if a > 5:  
    print("This number is big.")  
    print("This only gets printed if a is larger than 5.")  
print("This always gets printed.")
```

Condition

Body

In Python, the syntax of an if statement is the keyword `if` followed by the condition, followed by a colon (`:`), followed by the body. In Python, the body **MUST** be indented (put a TAB or 4 spaces at the start of the line).

If statements

```
a = int(input())

if a == 1:
    print("a is equal to 1")

if a > 5:
    print("a is greater than 5")

if a < 2:
    print("a is less than 2")
```

```
12
a is greater than 5
1
a is equal to 1
a is less than 2
-6
a is less than 2
2
7
a is greater than 5
```

If, elif, else

Here is a program that reads in the current temperature in Celsius, and outputs what state water would be in at that temperature.

```
tempr = int(input())

if tempr >= 100:
    print("It is very hot.")
    print("Water would boil at this temperature.")
elif tempr >= 0:
    print("The temperature is somewhat reasonable.")
    print("Water would turn to liquid.")
elif tempr >= -273:
    print("It is cold.")
    print("Water freezes at this point.")
else:
    print("This is impossible.")
```

Comparison and logical operators

`==` Returns true if the two operands are equal.

`!=` Returns true if the two operands are not equal.

`<` Returns true if the first operand is less than the second.

`>` Returns true if the first operand is greater than the second.

`<=` Returns true if the first operand is less than or equal to the second.

`>=` Returns true if the first operand is greater than or equal to the second.

`and` Returns true if both operands are true.

`or` Returns true if at least one of the operands is true.

`not` Returns true if its single operand is false.

Booleans

A boolean variable can be either True or False.

```
a = True
b = False
c = 3 < 5

print(a)
print(b)
print(c)

if b or c:
    print("At least one of b and c is True")
```

While loop

A while loop keeps executing the code in its body while the condition is true.

```
n = int(input())

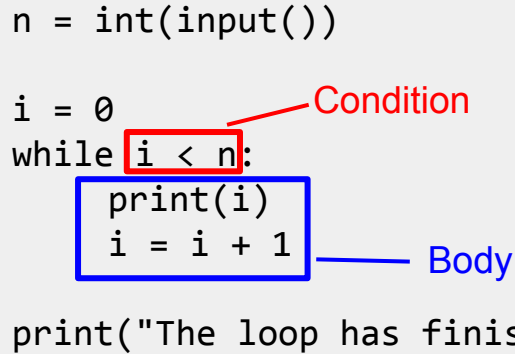
i = 0
while i < n:
    print(i)
    i = i + 1

print("The loop has finished executing.")
```


While loop

A while loop keeps executing the code in its body while the condition is true.

```
n = int(input())  
  
i = 0  
while i < n:  
    print(i)  
    i = i + 1  
  
print("The loop has finished executing.")
```



Condition

Body

A while loop has very similar syntax to an if statement. It is the keyword `while` followed by a colon followed by the body (which is indented).

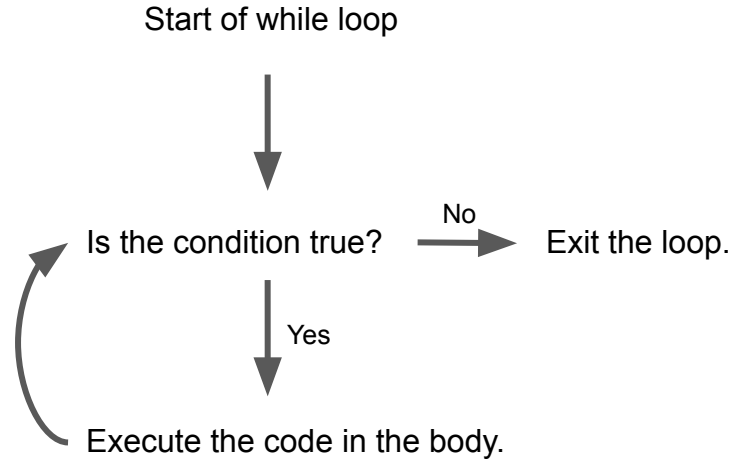
While loop

A while loop keeps executing the code in its body while the condition is true.

```
n = int(input())

i = 0
while i < n:
    print(i)
    i = i + 1

print("The loop has finished executing.")
```



More assignment operators

`a += b` is the same as `a = a + b`

`a -= b` is the same as `a = a - b`

`a *= b` is the same as `a = a * b`

`a /= b` is the same as `a = a / b`

`a //= b` is the same as `a = a // b`

`a %= b` is the same as `a = a % b`

`a **= b` is the same as `a = a ** b`

For example, `i = i + 1` can be rewritten as `i += 1`

Arrays

You can use an array to store a list of elements.

```
arr = [100, 34, 65, 123, -74]
```

This will create an array named `arr` which stores 5 integers.

We can use square brackets to access elements of the array. Note that arrays are 0-indexed, meaning that their first element has index zero.

```
print(arr[0])  
print(arr[1])  
print(arr[2])  
print(arr[3])  
print(arr[4])
```

```
100  
34  
65  
123  
-74
```

Arrays

You can also modify the elements of an array with the same syntax as that of variables.

```
arr = [100, 34, 65, 123, -74]

print(arr[2])
arr[2] = 13
print(arr[2])
arr[3] = arr[0] + arr[2]
print(arr[3])
```

What do you think this will output?

Arrays

You can also modify the elements of an array with the same syntax as that of variables.

```
arr = [100, 34, 65, 123, -74]

print(arr[2])
arr[2] = 13
print(arr[2])
arr[3] = arr[0] + arr[2]
print(arr[3])
```

65

13

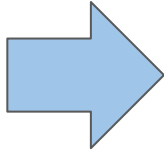
113

What do you think this will output?

What's the point of arrays?

You can put variables inside the square brackets.

```
print(arr[0])  
print(arr[1])  
print(arr[2])  
print(arr[3])  
print(arr[4])
```

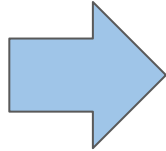


```
i = 0  
while i < 5:  
    print(arr[i])  
    i = i + 1
```

What's the point of arrays?

You can put variables inside the square brackets.

```
print(arr[0])
print(arr[1])
print(arr[2])
print(arr[3])
print(arr[4])
print(arr[5])
print(arr[6])
print(arr[7])
print(arr[8])
print(arr[9])
print(arr[10])
print(arr[11])
...
print(arr[999])
```



```
i = 0
while i < 1000:
    print(arr[i])
    i = i + 1
```


How would you even have such a large array?

In Python, you can use the multiplication operator on arrays.

```
arr1 = [3] * 4  
arr2 = [6] * 8  
arr3 = [1, 2, 3] * 5  
arr4 = arr1 * 2
```

```
print(arr1)  
print(arr2)  
print(arr3)  
print(arr4)
```

```
[3, 3, 3, 3]  
[6, 6, 6, 6, 6, 6, 6, 6]  
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]  
[3, 3, 3, 3, 3, 3, 3, 3]
```

How would you even have such a large array?

In Python, you can use the multiplication operator on arrays.

```
arr1 = [3] * 4  
arr2 = [6] * 8  
arr3 = [1, 2, 3] * 5  
arr4 = arr1 * 2
```

Commonly used

Rarely used

```
print(arr1)  
print(arr2)  
print(arr3)  
print(arr4)
```

```
[3, 3, 3, 3]  
[6, 6, 6, 6, 6, 6, 6, 6]  
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]  
[3, 3, 3, 3, 3, 3, 3, 3]
```

Array length

Python's `len` function returns the length of an array.

```
arr = [100, 34, 65, 123, -74]
```

```
i = 0
while i < len(arr):
    print(arr[i])
    i = i + 1
```

```
100
34
65
123
-74
```

Iterating through an array

```
arr = [0] * 100

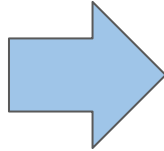
i = 0
while i < len(arr):
    print(arr[i])
    i = i + 1
```

This will print 100 0s.

For loops

For loops in Python allow you to iterate over an array much more conveniently.

```
i = 0
while i < len(arr):
    a = arr[i]
    print(a)
    i = i + 1
```



```
for a in arr:
    print(a)
```

Range function

The `range` function returns an array-like object that contains a range of numbers. The biggest difference between the object returned by `range` and a normal array is that you can not modify the object returned by `range`.

For loops and range function

`range(n)` returns a sequence containing all of the integers from 0 inclusive to n exclusive.

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

Is basically
equivalent to

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:  
    print(i)
```

This would print all numbers between 0 inclusive and 10 exclusive.

In math and programming (when talking about programming, not in actual code), we write this range as $[0, 10)$. Square brackets mean inclusive while parenthesis mean exclusive.

For loops and range function

If you pass two arguments to the range function, a, b, it will return the sequence of integers from a inclusive to b exclusive.

```
for i in range(3, 10):  
    print(i)
```

Will print all the integers in [3, 10).

For loops and range function

If you pass three arguments to the range function, a , b , c , it will return every c^{th} integer in the range $[a, b)$, starting at a . The third argument is often referred to as the “step”, as in the size of the step between two numbers in the returned sequence.

```
for i in range(3, 10, 2):  
    print(i)
```

Will print integers in $[3, 10)$, but instead of each number being 1 larger than the previous one, each number will be 2 larger than the previous one. For example, the above code will print the numbers 3, 5, 7, 9.

I do not have any fancy math notation for this :(

Practice Exercises

Exercise 1

Write a program that reads in a single integer, and outputs fizz if it is divisible by 5, buzz if it is divisible by 7, fizzbuzz if it is divisible by both 5 and 7, and the number itself otherwise.

Hint: you can use the % operator to check whether one number is divisible by another.

Note: this question is a slightly easier version of the one on the survey. That question required you to do this for every number in between 1 and N.

Exercise 2

Write a program that reads in a single integer N , and outputs the first N perfect squares (starting from 1^2 and including N^2).

Exercise 3

Write a program that reads in an integer and counts how many decimal digits it has.

Note: do not use strings. This is solvable with only stuff we learned in our first two meetings.

Exercise 4

Read in a single integer and output all of its factors in increasing order.

Exercise 5

Write a program that reads in a single integer N , and for each integer i in $[1, N]$, outputs `i is prime` if the number is prime and `i is not prime` otherwise.