Python

Object-oriented programming

C

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

```
string answer = get_string("what's
    your name? ");
printf ("hello, %s\n", answer);
```

≠

Python

```
printf("hello, world")
```

The new line here is automatically by default

```
answer = get_string("what's
    your name? ")
print("hello, " + answer)
```

There's no type, like "string", which even though that was a type in C50, every other variable in C did we use int or string or float, or Bool or something else.

In Python, there are still going to be data types, but you, the programmer, don't have to bother telling the computer what types you're using.

→ the computer is going to be smart enough, the language, to just figure it out from context.

Get_string is going to be a function we'll use today, which comes from a Python version of the CS50 library. But we'll also start to take off those training wheels, so that you'll see how to do things without any CS50 libraries going forward, using a ≠ function instead.

⊕ Concatenation ⊕

⊕ is answer to whatever is quoted here

```
print("hello, " + answer)
```

Python should assume that anything inside of curly braces inside of the string should be interpolated; which is a fancy term saying, substitute the value of any variables therein.

```
print(f"hello, {answer}")
```

allows you to plug things in.

variable declared on the 1st time

`int counter = 0;` ≠ `counter = 0`

You don't need to mention the type, just like before with string and you don't need a semicolon.

Simpler.

If you want a variable, just write it and set it equal to some value.

Increment → Almost the same

```
counter = counter + 1;
counter += 1;
i++;
```

→

```
counter = counter + 1
counter += 1
it doesn't exist in Python!
i = i + 1
```

Condition LESS THE TYPE

```
if (x < y)
{
    printf ("x is less than y\n");
}
else
{
    printf ("x is not less than y\n");
}
```

```
if x < y:
    print ("x is less than y")

IDENTATION IN PYTHON IS IMPORTANT

else:
    print ("x is not less than y")
```

```
if (x<y)
{
    printf ("x is less than y\n");
}
else if (x>y)
{
    printf ("x is greater than y\n");
}
else
{
    printf ("x is not equal to y\n");
}
```


```
if x < y:
    print ("x is less than y")
elif x > y:
    print ("x is greater than y")
else:
    printf ("x is equal to y")
```

(1) Is Python sensitive to spaces and where they go?
Sometimes no, sometimes yes.
Formal style conventions

```c
while (true)
{
    printf("meow\n");
}
```

```python
while True:
    print("meow")
```

Capitalized because True and False are boolean values in Python!

```c
int i = 0;
while (i < 3)
{
    printf("meow\n");
    i++;
}
```

```python
i = 0
while i < 3:
    print("meow")
    i += 1
```

```c
for (int i = 0; i < 3; i++)
{
    printf("meow\n");
}
```

```python
for i in [0, 1, 2]:
    print("hello, world")
```

(array)

range(n) = 0, 1, 2, 3, ..., n-2, n-1

```python
for i in range(20):
    print("doh")
```

Lists in python are more like link lists than they are arrays.

If you don't know how many times you want to loop or iterate, you can't really create a hard-coded list like [0, 1, 2].

```
for i in [0,1,2]:
        =
for i in range(3):
```

} it's the same :)

Types

PYTHON

bool	bool
char	
double	
float	float
int	int
long	
string	str
...	...

Integer overflow is no longer going to be an issue.
→ Float it's a problem that remains.

range	dict
list	set
tuple	...

Functions

get_char
get_double
get_float
get_int
get_long
get_string
"

get_float
get_int

get_string

Library

#include <cs50.h> → import cs50

there aren't header files anymore!
it's library

from cs50 import
get_float, get_int,
get_string

← from cs50 import get_float
from cs50 import get_int
from cs50 import get_string

Compile

make hello
./hello

→ python hello.py

source code → interpreter →

passed into an interpreter, not a compiler

blur.py

```
from PIL import Image, ImageFilter
                          → function
before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BoxBlur(1))
after.save("out.bmp")
```

variable → struct or object

((In object-oriented programming, in a language like Python, you can encapsulate not just values, but also functionality.))

Inside of the image library, there's a function called open, and it takes an argument

Language C

```
#include <unistd.h>

void ft_write_number(int number)
{
    if (number > 9)
        ft_write_number(number/10);
    write(1, &"0123456789"[number % 10], 1);
}

int main(void)
{
    int number;

    number = 1;
    while (number <= 100)
    {
        if(number % 3 == 0 && number % 5 == 0)
            write(1, "fizzbuzz", 8);
        else if (number % 3 == 0)
            write(1, "fizz", 4);
        else if (number % 5 == 0)
            write(1, "buzz", 4);
        else
            ft_write_number(number);
        write(1, "\n", 1);
        number++;
    }
}
```

```
def writeNumber ( number)
    if number > 9:
        writeNumber print ("numéro =", (n//10)
    else:
        print ("numéro =", (n%10)

number = 1
while number <= 100
    if (number %3==0) and (number %5==0):
        print ("fizzbuzz")
    elif (number %3 == 0):
        print ("fizz")
    elif (number %5 == 0):
        print ("buzz")
    else:
        print ("o numéro é ", number)
    numer = number + 1
```

Operadores

exponenciação — **
multiplicação — *, //, /, %
adição — +, -
relacional — ==, !=, <=, >=, >, <

lógico — not
lógico — and
lógico — or

BLURRING AN IMAGE

from PIL import Image, ImageFilter

```
before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BoxBlur(10))
after.save("out.bmp")
```

It just saves the file

The argument here happens to be image.boxblur(1), which itself is a function, but it just returns the filter to use.

So, instead of using fopen and fwrite, you just say .save and that does all of that messy work for you.

So it's just four lines of code total?

Yes. So the library is doing ALL of the same kind of legwork that you all did for the assignment, but it's encapsulated it all into a single library, that you can then use instead.

Those of you who might have been feeling more comfortable, might have done a little something like this:

edges.py

PIL = Pillow library

```
from PIL import Image, ImageFilter

before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.FIND_EDGES)
after.save("out.bmp")
```

FIND_EDGES → Is like a constant, if you will defined inside of this library for us.

```
$ code edges.py
$ python edges.py
$ code out.bmp
```

Amazing!!! So again, suggesting the power of using a language that's better optimized for the task at hand. The risk of really making folks sad, let's

go ahead and re-implement, if we could, problem set five, real quickly here.

~ " ~

Dictionary.py

I have a C version, just from problem set five, wherein you implemented a spell checker, loading 100000+ words into memory.

And then you kept track of just how much time and memory it took. *

That probably took a while, implementing all of those functions in Dictionary.py.

Let me instead now go into a new file and let me stipulate, for the sake of discussion, that we already wrote in advance, speller.py, which corresponds to speller.c.

So the onus on us right now is only to implement Speller, Dictionary.py; All right, so I'm going to go ahead and define a few functions.

We're going to see now the syntax for defining func

tions in Python.

dictionary.py

words = ~~dict~~ set() } defining a variable

def check(word):
    ...

you don't specify the return type

You use the word Def instead to define

You still specify the name of the function and any arguments there too. But you omit any mention of types.

    if word.lower() in words:
        return True
    else:
        return False

→ little similar to fopen

def load(dictionary):
    file = open(dictionary, "r")
    for line in file:
        word = line.rstrip()
        words.add(word)
    file.close()
    return True

```
def size():
    return len(words)

def unload():
    return True
```

So I give you, in these functions, problem set five in Python; what are you getting for free, in a language like Python?

Well, encapsulated in this one line of code:

```
words = dict()
```

is much of what you wrote for problem set five, implementing your array for all of your letters of the alphabet or more, all of the linked list that you implemented to create chains, to store all of those words.

Set just allows it to handle duplicates, and it allows me to just throw things into it by literally using a function as simple as add.

And I'm going to make one other tweak here, because, when I'm checking a word, it's possible it might be given to me in uppercase or capitalized.

It's not going to necessarily come in at the same lowercase format that my dictionary did.

I can force every word to LOWERCASE by using (word.lower)

And I don't have to do it character for character, I can do the whole damn string at once, by first saying word.lower.

All right, let me go ahead and open up a terminal window here:

```
$ make speller
$ ./speller texts/holmes.txt
```

```
$ python speller.py
texts/holmes.txt
```

So the syntax is a little different at the command prompt, on the left I have to compile the code, with make, and then run it with ./speller.

On the right, I don't need to compile it, but I do need to use the interpreter.

It was more faster than that

Python is more comfortable for the programmer, but C is better for the user.

If you have a very large data set, you might want to optimize your code to be as fast and performant as it can be, especially if you're running that code again and again.

You might want to have someone smart take a language like C and write it at a very low level and it's going to be painful. They're going to have bugs. They're going to have to deal with memory management and the like.

→ But if and when it works correctly, it's going to be much faster, it would seem.

Could you just compile Python code?

Yes, absolutely, this idea of compiling code or interpreting code is not native to the language itself. It tends to be native to the conventions that we humans use.

→ You could actually write an interpreter for C that would read it top to bottom, left to right, converting it to, on the fly, something the computer understands, but historically, that's not been the case.

Python is compiling the code, technically, not into 0's and 1's, into something called byte code, which is this intermediate step that just doesn't take as much time as it

would to recompile the whole thing.

This is an area of research for computer scientists working in programming languages, to improve these kinds of paradigms.

↳ For the programmer, it's just much easier to run the code and not worry about the stupid second step of compiling it all the time. It's literally half as many steps for me, the human.

I didn't even need to implement main!

↳ In C you have to have that to kick-start the entire process of actually running your code.

And in fact, if you were missing main, as you might have experienced if you accidentally compiled helpers.c instead of the file that contained main, you would have seen a compiler error.

In py you can just jump right in, start programming and boom! you're good to go. You don't need the added overhead & complexity of a main function.

1º)

```
answer = input ("what's your name?")
print (f"hello, {answer}")
```

2º) from cs50 import get_string

```
answer = get_string ("what's your name?")
print (f"hello, {answer}")
```

3º) from cs50 import get_string

```
answer = get_string ("what's your name?")
print ("hello,", answer)
```