Programming Workshops

Principles of Computing

Dr. Joseph Walton-Rivers

Week 1

Today's Topics

- Variables
- Functions (using)
- Branching

But first...

Question
How did you get from arriving on campus this morning to this room?

Abstraction!

- Did these answers seem correct to you?

Abstraction!

- Did these answers seem correct to you?
- How did they differ?

Abstraction!

- Did these answers seem correct to you?
- How did they differ?
- (either I got lucky and you did this without thinking, or I added my own description to the end)

Abstraction!

- Did these answers seem correct to you?
- How did they differ?
- (either I got lucky and you did this without thinking, or I added my own description to the end)
- Some seemed more **detailed** than others

Abstraction!

- Did these answers seem correct to you?
- How did they differ?
- (either I got lucky and you did this without thinking, or I added my own description to the end)
- Some seemed more **detailed** than others
- We call this ’detail’ *abstraction* in programming

Turing Machines

- The computational model I described in previous session was *correct*
- But it also was very ‘low-level’, which makes it a poor choice for day-to-day programming
- We abstract this away by programming in a higher-level of abstraction
- So which level of abstraction is the *right* level of abstraction?

Turing Machines

- The computational model I described in previous session was *correct*
- But it also was very ‘low-level’, which makes it a poor choice for day-to-day programming
- We abstract this away by programming in a higher-level of abstraction
- So which level of abstraction is the *right* level of abstraction?
- The one that gets the job done
- Often, solving problems requires thinking across these levels of abstraction

Variables

Variables

- Variables are how we can store data that *might* change
    - In constrast to a literal - 42, 'a', or "Hello" (debated – not for now)
    - And constants - "variables that can't change"
- Variables have a *type* and a *name*
- In C-Sharp, when you declare a variable, you write the *type* followed by the name
- When you *access* a variable you just use the name
- We'll talk more about *types* in a later session

Types: Crash course

Common variable types:

Type	Value	Example
int	whole numbers	42
float	numbers with a factional component	3.14
string	sequences of text	”Hello world”
bool	true or false	true

An Example - declaring a variable

```
int myValue;
```

- What is the type?

An Example - declaring a variable

```
int myValue;
```

- What is the type? int
- What is the name?

An Example - declaring a variable

```
int myValue;
```

- What is the type? int
- What is the name? myValue
- if I wanted to make the type a *float* what would I change?

An Example - declaring a variable

```
int myValue;
```

- What is the type? int
- What is the name? myValue
- if I wanted to make the type a *float* what would I change? int becomes float

An Example

```
myValue = 42;
```

- What is the name of the variable on this line?

An Example

```
myValue = 42;
```

- What is the name of the variable on this line? myvalue
- I’m assigning an integer literal to this. What is that value?

An Example

```
myValue = 42;
```

- What is the name of the variable on this line? myvalue
- I’m assigning an integer literal to this. What is that value? 42

Functions

Functions as abstraction

- Having to manually write out how to do something over and over again from the lowest-possible level of detail gets tedious quickly
- Instead, it’s useful to treat ‘units of work’ as black boxes which we can use
- Sometimes these black boxes have *side-effects* this makes Dijkstra, unit testers (and Haskell programmers) sad

A function

- We can think of a (pure) function as something that:
    - Has a name (WriteLine)
    - Takes 0 or more arguments, which have **both**:
        - a **type** (String, Int) - we'll be looking at this in the next worksheet
        - (most of the time) a **name**
    - Can return a value - we don't give this a name just a type (int, float)
- Pure means has no side-effects, the functions we're using today will have side effects, as will almost all the things you do in game engines/robotics
    - We call functions that 'live inside' classes **methods**
    - Some programmers might call functions that live outside classes 'free functions'

WriteLine

Let’s take a look at a function called WriteLine (in the Console class)

```
static void WriteLine (string format, object? arg0, object? arg1);
```

- What is this function’s name?

WriteLine

Let's take a look at a function called WriteLine (in the Console class)

```
static void WriteLine (string format, object? arg0, object? arg1);
```

- What is this function's name? WriteLine
- What is this function's arguments?

WriteLine

Let's take a look at a function called WriteLine (in the Console class)

```
static void WriteLine (string format, object? arg0, object? arg1);
```

- What is this function's name? WriteLine
- What is this function's arguments? format, arg0, arg1
- What is this function's return type?

WriteLine

Let’s take a look at a function called WriteLine (in the Console class)

```
static void WriteLine (string format, object? arg0, object? arg1);
```

- What is this function’s name? WriteLine
- What is this function’s arguments? format, arg0, arg1
- What is this function’s return type? void (nothing)
- Also, static is here, that means we don’t work on an *instance* of its class. Don’t worry about that for now, we’ll get to that later (and you’ll see OO stuff with Matt)

Documentation

- That doens’t actually tell us what the function *does*
- This is why ’my code is self documenting’ is rarely true by the way
    - As programmers technical writing is a skill we need to master for our ‘other audiance’
- Writes the specified data, followed by the current line terminator, to the **standard output** stream.

Documentation

- That doens’t actually tell us what the function *does*
- This is why ’my code is self documenting’ is rarely true by the way
    - As programmers technical writing is a skill we need to master for our ‘other audiance’
- Writes the specified data, followed by the current line terminator, to the **standard output** stream.
- Ah! that was something Joe mentioned in the session the other day!

Documentation

- That doens’t actually tell us what the function *does*
- This is why ’my code is self documenting’ is rarely true by the way
    - As programmers technical writing is a skill we need to master for our ‘other audiance’
- Writes the specified data, followed by the current line terminator, to the **standard output** stream.
- Ah! that was something Joe mentioned in the session the other day!
- This is a function that prints things to the screen!

Function as contract

Ok, so here's what we know from the function signature:

- This function is something we can give three 'things' (arguments) to:
    - A string called format (whatever that is)
    - An ' object? ' called arg0
    - An ' object? ' called arg1
- What's the next thing we would like to know?

Function as contract

Ok, so here’s what we know from the function signature:

- This function is something we can give three ‘things’ (arguments) to:
    - A string called format (whatever that is)
    - An ‘ object? ‘ called arg0
    - An ‘ object? ‘ called arg1
- What’s the next thing we would like to know?
    - object is c-sharp for ”I don’t care, give me anything”
    - ‘ ? ‘ is c-sharp for ’or possibly null’
- We also know this function will return nothing (void)

Calling the function

```
Console.WriteLine(”Hello, World!”, null, null);
```

Calling the function

Console.WriteLine(”Hello, World!”, null, null);

There is also a single and multi-argument versions of this function. I wanted to make it clear what was an argument and what was a return type by providing different numbers of them in my ealier slide.

Calling the function

Console.WriteLine(”Hello, World!”, null, null);

There is also a single and multi-argument versions of this function. I wanted to make it clear what was an argument and what was a return type by providing different numbers of them in my ealier slide.

Passing null to this function twice is silly...

Calling the function

```
Console.WriteLine(”Hello, World!”);
```

- Because C-Sharp is Object-Oriented, all functions are tied to a class (this might be strange if you’re used to Python)

Calling the function

```
Console.WriteLine(”Hello, World!”);
```

- Because C-Sharp is Object-Oriented, all functions are tied to a class (this might be strange if you’re used to Python)
- Perhaps not to the team writing full-on python classes in the robot oympaid last week
- The class where ‘WriteLine‘ ’lives’ in C-Sharp is called ‘Console‘
- This class also handles input

Branching (conditionals)

Branching

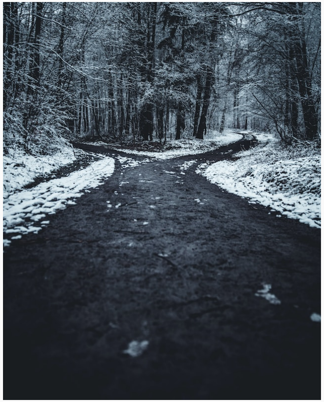
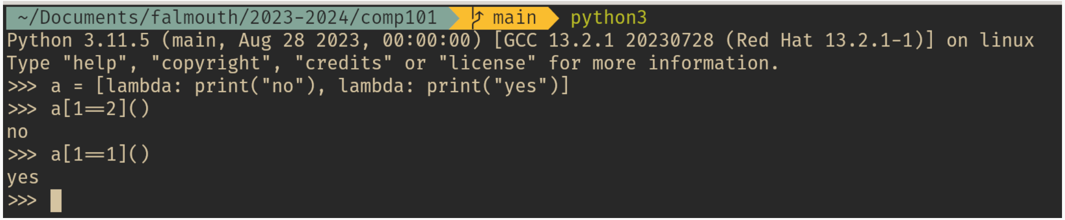


Photo by Oliver Roos on Unsplash.

Two Paths

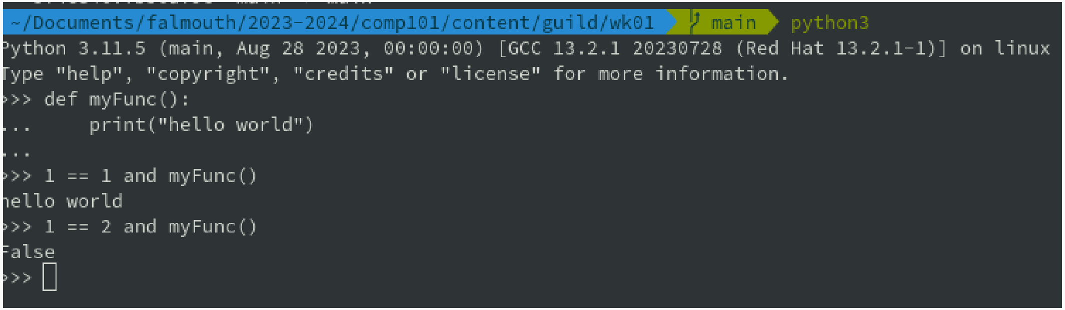
- Sometimes we want to do *something* only if something else is true
- We do this using ‘if‘ and ‘else‘
- You can do this without those constructs, but then you’re in wacky golfing territory.
- And no one wants to see that...

Ok, fine...



```
 ~/Documents/falmouth/2023-2024/comp101  main  python3
Python 3.11.5 (main, Aug 28 2023, 00:00:00) [GCC 13.2.1 20230728 (Red Hat 13.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = [lambda: print("no"), lambda: print("yes")]
>>> a[1==2]()
no
>>> a[1==1]()
yes
>>> 
```

Also short-circuit operators...



```
~/Documents/falmouth/2023-2024/comp101/content/guild/wk01  main  python3
Python 3.11.5 (main, Aug 28 2023, 00:00:00) [GCC 13.2.1 20230728 (Red Hat 13.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def myFunc():
...     print("hello world")
...
>>> 1 == 1 and myFunc()
hello world
>>> 1 == 2 and myFunc()
False
>>> 
```

If statement

- We have a condition ‘if (a == 42)‘
- We have code that will run *only* if this is the case
- We (optionally) have code that will run if it is not the case (*else*)

In code

```
if ( condition ) {
// Do things if true
} else {
//Do things if false
}
```

Loops

- We’ll talk more about iteration next week
- For the extention task at the end, you might find this bit useful

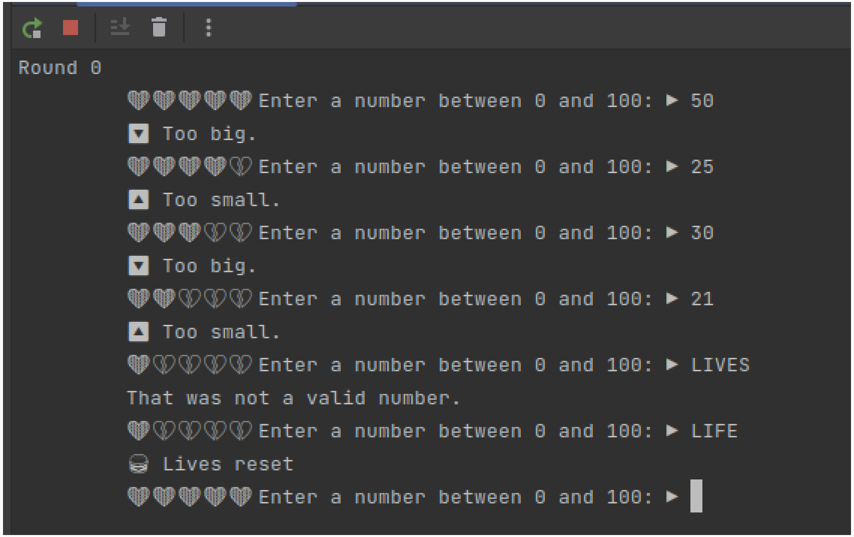
(advanced) Repeating

- Another form of branching is the loop
- Simplest form: while
- When a condition is *true* execute the code in the body of the statement
- Then jump back to the condition
- Make sure the condition can change in the loop, else it will loop forever!

In Code

```
int myValue = 0;
while ( myValue < 4 ) {
// Do things here
myValue++;
}
```

Guessing Game



```
Round 0
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ 50
          🔽 Too big.
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ 25
          🔼 Too small.
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ 30
          🔽 Too big.
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ 21
          🔼 Too small.
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ LIVES
          That was not a valid number.
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ LIFE
          🍔 Lives reset
          💔💔💔💔💔 Enter a number between 0 and 100: ▶ 
```