Introduction to Pygame

Kieran Vickers

Starter

Where's the mistake?

Find the syntax errors

```
item_costs = [2.5, 3.75, 1.99, 5.00]  # £
item_quantities = [2, 4, 5, 6]
total_cost = 0  # £
credit_limit = 100  # £
if not len(item_costs) == len(item_quantities):
    raise AssertionError("Invalid Inputs")
for index, item_cost in enumerate(item_costs):
    quantity = item_quantities(index)
    subtotal = item_costs * quantity
    total_cost = item_cost
if total_cost =< credit_limit:
    print("Sale approved, £" + str(total_cost))
else:
    print("You cannot afford this.")
```

Mistakes

There are two syntax errors: - `item_quantities(index)` is a syntax error, to take an item from a list use `[ & ]` –> `item_quantities[index]` - `total_cost =< credit_limit` is a syntax error –> `total_cost <= credit_limit`

Where's the mistake?

Find the logical error

```
item_costs = [2.5, 3.75, 1.99, 5.00]  # £
item_quantities = (2, 4, 5, 6)
total_cost = 0  # £
credit_limit = 100  # £
if not len(item_costs) == len(item_quantities):
    raise AssertionError("Invalid Inputs")
for index, item_cost in enumerate(item_costs):
    quantity = item_quantities[index]
    subtotal = item_costs * quantity
    total_cost = item_cost
if total_cost <= credit_limit:
    print("Sale approved, £" + str(total_cost))
else:
    print("You cannot afford this.")
```

Mistakes

There is one logic error: - `total_cost = item_cost` is a logical error - `total_cost += item_cost`

What we aim to cover

What we aim to cover

- Cover some Python basics ready for object oriented programming
- First taste of pygame
- A quick look at an example game in pygame
- This slide deck will likely take more than one workshop

You will need

- . . . A broswer open on a search engine
- . . . IDLE open (try `import pygame` now)
- . . . to be willing to ask questions

Let's Talk Python

Let's Talk Python

We need to ensure we are all working on the same page. Python is a fantastic language because you can have a working piece of code in seconds, but it has enough features to be incredibly powerful and fast. As a result, there are many many many ways to do the same thing, and always something new to learn.

Variables, Constants and Functions

Python requires you to set a variable or constant before it is used. In Python it is common to use `ALL_UPPER_CASE_WITH_UNDERSCORES` for constants, and `all_lower_case_with_underscores` for variables and functions, although there is always debate.

Lists

Lists are defined using

```
my_list = ["hello", "world"]
my_other_list = [0, 1, 2]
my_empty_list = list()
my_other_empty_list = []
both_lists = my_list + my_other_list
both_lists.append(False)
```

What is the value of `both_lists`?
It is *convention* to only store one type in lists, but Python **really** doesn't care.

Tuples

Tuples are defined using

```
my_tuple = ("hello", 5, False)
my_empty_tuple = tuple()
my_other_empty_tuple = ()
```

What is the value of `my_tuple[1]`? With tuples you cannot add or remove from them once defined, it is useful to store a fixed collection of things (such as 2D coordinates).

Dicts

Dicts are defined using

```
my_tuple = {"name": "James", "age": 12, True: False}
my_empty_tuple = dict()
my_other_empty_tuple = {}
my_tuple["favourite_food"] = "Apples"
```

Dicts, or dictionaries, map a key to a value, and do not store any sort of order. Keys **must** be unique, and by convention, keys *should* of one type, strings, (and in other languages, so should values) but again, Python **really** doesn't care.

Accessing Dicts

The value of a dict can be accessed directly two ways

```
my_tuple = {"name": "James", "age": 12}
# Method one
print(my_tuple["name"])
print(my_tuple["favourite_food"])
# Method two
print(my_tuple.get("name", "some_default_value"))
print(my_tuple.get("favourite_food", None))
```

Types

Python is “duck-typed”.

> *If it walks like a duck, and talks like a duck, it is a duck.*

. . . in other words, unless something breaks, let’s just guess what type something is.
Python does this quite well (compared to JavaScript), and also allows you to specify types if you *really* want to.

Types II

Basic/built-in types are: `str`, `float`, `boolean`, `int`, `list`, `tuple`, `dict`, also `builtin_function_or_method`, `function`, `complex` and `type`.

Selection

Python has one selection: `if expression:`, which can be followed by `elifs` and `else`.

```
my_variable = "duck"
my_list = [1, 2, 3]
if "hello" and 5 and my_variable is "duck" \
        and my_list == [1, 2, 3] and 2 in my_list \
        and "a" not in my_variable and not []:
    print("This is true?!?")
```

Expressions I

Any object (including of built-in types) evaluate to true or false. The "empty" value of the built-in types are false, all others are true.

So, expressions can be: a boolean value, a boolean expressions, an object (which therefore can be evaluated to true or false), or a function that returns any of the prior.

These can all be combined using `and`, `or` and `not`.

Expressions II

Individual boolean expressions can use: `<`, `>`, `<=`, `>=`, `==`, `!=`, `in`, `not in`, `is`, `is not`.

Task: what is the difference between `==` and `is`?

`is` vs `==`

An `is` expression evaluates to True if two variables point to the same (identical) object.

An == expression evaluates to True if the objects referred to by the variables are equal (have the same contents).

is vs == II

So, what is the three statements outputted here?

```
list_one = [1, 2, 3]
list_two = list_one
list_three = list(list_one)
print(list_one == list_two == list_three)
print(list_one is list_two)
print(list_one is list_three)
```

Iteration

Python has two loops: `while expression`, which can take any expression an `if` can.
The second is `for variable(s) in iterable`.

```
for i in (1, 3, 5, 7):
    print(i)
for j in range(10):
    print(j)
for k in "london":
    print(k)
for index, value in enumerate(["i", "love", "pygame"]):
    print(index, "-->", value)
positions = [(0, 1), (5, 2), (8, 1)]
for x, y in positions:
    print(x, y)
```

Iterables

- strings, lists, tuples,
- dicts (which iterates over the keys, somewhat randomly (why?)),
- `dict.values()` (which iterates over the values),
- `dict.items()` (which iterates over (key, value) pairs),

Iterables II

- `range([start], end, [step])` (which iterates over a list of numbers),
- `enumerate(iterable)` (which iterates over (index, value) pairs, see starter),
- A function that `yields` *
- An object with `next()` and `iter()` methods *

Object Oriented Programming

```
class Animal:
    def __init__(self, name, number_of_legs):
        self.name = name
        self.number_of_legs = number_of_legs

    def greeting(self):
        return f"Hello, {self.name}"
```

A PyGame Game

A PyGame Game

If you go to
`tinyurl.com/kj-yr12-ex1`
and save this on your own system, you should be able to run it.

Run it

Run the pygame as you would any other Python file.
Don't look at the code yet, but think how it might work?

A REALLY SIMPLE PyGame "Game"

If you go to
`tinyurl.com/kj-yr12-ex0`
and save this on your own system, you should be able to run it.

Compare the games

The second example uses 18 lines of code, what does it do?
The first example uses 71 lines of code, how much more does it do?

An Object-Oriented Game

One more for you to download, go to
`https://tinyurl.com/kj-yr12-ex2`
and save this on your own system, you should be able to run it.

Compare the games

The third example uses 91 lines of code, the first example uses 71 lines of code, what is the difference between running them?

Look at the code

They're the same?!
Compare the first file (`example1.py`) and the third file (`example2.py`).
Which do you prefer? Which is easier to read? Which would you rather have to fix?
Which has the most repetition?

Let's Write

Let's write something

Look again at `example1.py`. Make changes to it:

- Can you change the colours?
- Can you change the speed the bullets fly?
- Can you change the number of bullets on the screen?

Blank Slate

Look at how *example2.py* works, and see if you can make a well documented (commented) "blank slate" program, similar to *example0.py* but object oriented.
It should display just a blank screen (with a constant for background colour).

See you next session!

See you in two weeks time!

Next session:

- Object oriented refresher
- Inheritance and abstracts
- The event loop
- More OO pygame