PARTIAL FUNCTIONS

Reducing Function Arguments

```
def my_func(a, b, c):
    print(a, b, c)
```

```
def fn(b, c):
    return my_func(10, b, c)
```

```
fn(20, 30)    → 10, 20, 30
```

```
f = lambda b, c: my_func(10, b, c)
```

```
f(20, 30)     → 10, 20, 30
```

```
from functools import partial

f = partial(my_func, 10)

f(20, 30)     → 10, 20, 30
```

Handling more complex arguments

```
def my_func(a, b, *args, k1, k2, **kwargs):
    print(a, b, args, k1, k2, kwargs)
```

```
def f(b, *args, k2, **kwargs):
    return my_func(10, b, *args, k1='a', k2=k2, **kwargs)
```

```
f = partial(my_func, 10, k1='a')
```

Handling more complex arguments

```
def pow(base, exponent):
    return base ** exponent


square = partial(pow, exponent=2)
cube = partial(pow, exponent=3)
```

```
square(5)     → 25

cube(5)       → 75

cube(base=5)  → 75
```

!! `square(5, exponent=3)` → 75

Beware!!

You can use variables when creating partials

but there arises a similar issue to argument default values

```
def my_func(a, b, c):
    print(a, b, c)


a = 10
f = partial(my_func, a)

f(20, 30)     → 10, 20, 30

a = 100

f(20, 30)     → 10, 20, 30
```

the memory address of 10 is baked in to the partial

a now points to a different memory address

but the partial still points to the original object (10)

If a is mutable (e.g. a list), then it's contents can be changed

Code