From Functions to Programs

Outline

Functions can Call Other Functions

We have already seen functions that call other functions to do their work:

```
def say_hello():
    print("hello, world!")
```

Functions Calling Functions

Of course, we can call our own functions too.

```
def squared_distance(p1, p2):
    (x1, y1) = p1
    (x2, y2) = p2
    dx = x2 - x1
    dy = y2 - y1
    return dx**2 + dy**2

def distance(p1, p2):
    sqdist = squared_distance(p1, p2)
    return math.sqrt(sqdist)
```

Designing Functions

We can use our own functions to reduce duplication in our code.

```
first_person = input("Person 1? ")
second_person = input("Person 2? ")
first_greeting = "hello, "+first_person
second_greeting = "hello, "+second_person
print(first_greeting)
print(second_greeting)
```

Designing Functions

Compare:

```
def greet(person):
    return "hello, "+person
print(greet(input("Person 1? ")))
print(greet(input("Person 2? ")))
```

Exercise

This is where we just ended up:

```
def greet(person):
    return "hello, "+person
print(greet(input("Person 1? ")))
print(greet(input("Person 2? ")))
```

Could duplication be reduced even further?

Is the result better or worse? Why?

Aesthetic: A Function Should Have One Job

- Python functions should be short
- Python functions should be simple

If we have a bug, we should be able to narrow down to exactly which function causes the problem.

Function Machines

Each function can be thought of as a little machine that processes inputs into outputs. The hardest part is *describing* the inputs and outputs in a way that is generally useful. For example:

- `greet` takes a person name as input and outputs an appropriate greeting
- `distance` takes a pair of 2D points and outputs their distance

This definition could be a great comment to put at the beginning of a function!

Breaking Down Complex Functions

Processing data may involve many steps. We have seen the use of variables to organize these steps, and often we can use functions as a more powerful tool towards the same end.

```
def transform_point(point, scale, translation):
    (px, py) = point
    (sx, sy) = scale
    (tx, ty) = translation
    return ((px+tx) * sx, (py+ty) * sy)
```

Breaking Down Complex Functions

This function is mostly OK, but is a bit inconvenient if we are using an identity in either scale or translation:

```
transform_point((10, 10), (1, 1), (200, 20))
# (1,1) means no change in scale
transform_point((5, 1), (2, 1), (0, 0))
# (0,0) means no translation
```

Breaking Down Complex Functions

Compare with this version:

```
def scale_point(point, scale):
    (px, py) = point
    (sx, sy) = scale
    return (px*sx, py*sy)
def translate_point(point, translation):
    (px, py) = point
    (tx, ty) = translation
    return (px+tx, py+ty)
def transform_point(point, scale, translation):
    return translate_point(scale_point(point, scale),
                           translation)
```

Breaking Down Complex Functions

Here we have dedicated functions for the basic operations and compose them to define `transform_point`.

This function also has different behavior from the previous version.

Specifically, it's *correct* (we should translate, then scale!).

Giving each function one role and *composing* functions makes the correct behavior easier to implement!

Exercise

Simplify:

```
def do_the_program():
    first_name = input("name? ")
    first_num = int(input("number? ")) * 2
    second_name = input("name? ")
    second_num = int(input("number? ")) * 2
    if first_num < second_num:
        swap = first_name
        first_name = second_name
        second_name = swap
        swap = first_num
        first_num = second_num
        second_num = swap
    print(first_name,first_num,second_name,second_num)
```

"Sans-IO"

It's tempting to write functions that grab data exactly when needed:

```
def greet():
    print("hello ",input("Name? "))


greet()
print("and also...")
greet()
```

But what if we wanted to change the greeting based on the name?
We'd need to rewrite the whole program.
Instead, we could separate the I/O from the computation...

"Sans-IO"

```
def greet(name):
    greeting = f"Hello, {name}!"
    if name == "Prof Osborn":
        greeting = f"Salutations, {name}!"
    return greeting

print(greet(input("Name? ")))
print("and also...")
print(greet(input("Name? ")))
```

This version puts all the input and output in one place, separating the way we get and present values from the way we process values.

Exercise

Here's a function that computes the area and perimeter of a rectangle. Rewrite it in sans-IO style (i.e., the `rect_props` function will need to take input parameters and return some output).

```
def rect_props():
    width = float(input("W? "))
    height = float(input("H? "))
    area = width*height
    perimeter = 2*(width+height)
    print(f"Area {area}, Perimeter {perimeter}")
```

Quiz