```
In [ ]:    1  from cs103 import *
           2
           3
```

CPSC 103 - Systematic Program Design

Module 04 Day 1

Rik Blok, with thanks to Ian Mitchell and Giulia Toti

Reminders

- this Wed-Fri: Module 4 Tutorial Attendance
- Wed: Module 3 (HtDD): Code Review
- Wed: Module 3 (HtDD): Tutorial Submission
- Wed: Module 1 (Intro): Tutorial Resubmission (optional)
- Mon: Module 5: Pre-Lecture Assignment
- Mon: Module 4 (Compound): Worksheet
- next Wed-Fri: Module 5 Week 1 Tutorial Attendance

See your Canvas calendar (https://canvas.ubc.ca/calendar) for details.

Aside: Back to intervals

Assume you are working with the following data type:

```
In [ ]:    1  Time = int # in range[0, 86400)
           2  # interp. seconds since midnight
           3
           4  T_MIDNIGHT = 0
           5  T_ONE_AM = 3600
           6  T_NOON = 43200
           7  T_END_OF_DAY = 86399
           8
           9  @typecheck
          10  # Template based on Atomic Non-Distinct
          11  def fn_for_time(t: Time) -> ...:
          12      return    (t)
```

iClicker question: Back to intervals

Let's say we design a function `is_it_time_yet(t: Time)` based on the data definition for `Time` above. What happens if our program calls `is_it_time_yet(99999)` ?

A. The program accepts the value and continues
B. The program generates an error
C. The program waits for a valid `Time`
D. The program terminates abruptly
E. Something else

> ▶ ℹ Hint (for your review after class)

Designing functions (HTD**F**) with intervals

When designing functions (HTD**F**) you will sometimes use intervals as function inputs or outputs. Here are some things to consider.

✅ Intervals as function inputs (parameters)

In this course, you **may assume** that interval values passed into your functions are always valid – that is, within their specified range. For example, you **don't** need to verify that the parameter `t` is in the range `[0, 86400)` here:

```python
@typecheck
def is_afternoon(t: Time) -> bool:
    ''' Returns True if the time `t` is afternoon, otherwise False. '''
    # return True # stub
    # if t < 0 or t >= 86400: # you are NOT REQUIRED to check this
    # range, so you can skip this
    # template from Time
    return t >= T_NOON
```

⚠️ Intervals as function outputs (return values)

In this course, you **must ensure** that interval values returned by your functions are always valid – that is, within their specified range. For example, you **must** check the value of `time_left` before returning it in the function below:

```python
@typecheck
def time_remaining(temperature: float, heat: HeatLevel) -> Time:
    ''' Returns time remaining for a pot to boil. '''
    # return 0 # stub
    # template from atomic non-distinct and HeatLevel
    time_left = (100 - temperature) / heat # arbitrary formula, details unimport
ant

    # At this point time_left could be outside of Time's allowed range.
    # Need to check range before returning value of type Time.
    if time_left < 0:
        time_left = 0
    elif time_left >= 86400:
        time_left = 86399
    return time_left
```

Data types

Last week:

- Simple atomic
- Interval
- Enumeration
- Optional

Q: What is an example of each?

Now:

- **Compound data (Module 4)**

Later:

- Arbitrary-sized (Module 5)

Module learning goals

By the end of this module, you will be able to:

- Identify problem domain information that should be represented as compound data.
- Understand and write `NamedTuple` definitions.
- Use the HtDD, and Data Driven Templates recipes with compound data.
- Design functions that take in and/or return compound data.

Compound data

Sometimes, the information to be represented has two or more values that naturally belong together:

- Preferred name, surname, and student ID of a student
- Title, artist, album, and duration of a song
- Title, year, and director of a movie
- ...

Compound data offers a way to handle that! It lets you create a data definition to represent different **attributes** of a single entity. For example, we could define a `Student` compound to store a student's preferred name, surname, and ID.

Example

We'll represent compound data with the `NamedTuple` data type. For example, let's say we wanted to represent a velocity, including a speed and direction:

```
from typing import NamedTuple
Velocity = NamedTuple('Velocity', [('speed', float),
                                   ('dir', int)]) # in range[0,359]
```

▼ ✅ Any number of fields ok

Alternatively, could have more fields. Notice the pattern:

```
Velocity = NamedTuple('Velocity', [('speed', float),
                                   ('dir', int), # in range[0,359]
                                   ('x_component', float),
                                   ('y_component', float)])
```

```
# interp. a velocity with its speed in m/s and direction
# as an angle in degrees with east=0 increasing counterclockwise

v1 = Velocity(9, 22)
v2 = Velocity(3.4, 180)

# template based on Compound
@typecheck
def fn_for_velocity(v: Velocity) -> ...:
    return ...(v.speed, v.dir)
```

Notice how the fields (`speed` and `dir`) are treated as parameters in the template's `return` statement, using "dot notation":

- A function we design might not use them all but the data template shows all available information
- Each field has a data type. Can be anything... simple atomic, interval, enumeration, even another compound(!), and more!

Glossary

- **tuple** - a set of attributes that belong together (e.g., `Velocity` has attributes `speed` and `dir`)
  - from the latin suffix *-uple*, e.g., co**uple**, tri**ple**, quadr**uple**, quin**tuple**, ...
- **field** - the name of an attribute (e.g., `speed` or `dir`)
- **value** - the data assigned to an attribute (e.g., `3.4` or `180`)
- **instance** - a specific realization of a data type (e.g., `Velocity(3.4, 180)`)

iClicker question: Name that tuple

You're designing a data definition for wall paint, with the following properties: colour, finish (glossy or flat), drying time, and base (oil or latex).

What should you choose for a name to give your new data type?

A. `Colour`
B. `Finish`
C. `DryingTime`
D. `Base`
E. Something else

> ► ℹ️ Hint (for your review after class)

Instances of compounds

- Create a new instance of a compound data type with name of type and list of values for arguments. E.g.,

  ```
  v2 = Velocity(3.4, 180)
  ```

Need...

- the right number of arguments, e.g. two,
- in the right order, e.g., `speed` before `dir` ,
- of the right type, e.g., `float` then `int # in range[0,359]`

(Analagous to argument requirements when calling functions.)

> ### ✅ Creating an instance of a compound data type with named attributes
>
> It is also possible to create an instance of compound data using "named attributes". For example
>
> ```
> v2 = Velocity(speed=3.4,dir=180)
> ```
>
> will produce an instance of Velocity with the same attribute values as the earlier example (which uses what are called "positional attributes"). When you use positional attributes, you must provide the values in exactly the same order as the attributes were defined. When you use named attributes, you can provide the values in any order; for example
>
> ```
> v2 = Velocity(dir=180, speed=3.4)
> ```
>
> produces an instance of Velocity with the same attribute values as the earlier examples. That said, you must use the exact names of the attributes (and those names, like everything in Python, are case sensitive).
>
> In both cases (named or positional attributes) you must provide values for **all** of the attributes.
>
> It is technically possible to mix positional and named attributes when creating an instance of a compound data type, but **do not do that**. It is way too likely to cause confusion for you or some other reader of your code.
>
> In CPSC 103 we will generally stick to using positional attributes during instance creation.

iClicker question: Short answer

Besides `preferred_name`, `surname`, and `id`, what's another field you might add to a `Student` compound data type? Enter the field name as a valid identifier in *snake_case*, e.g., `preferred_name`. [Set iClicker question type to "Short Answer".]

Cartesian coordinates

Let's create a data type to work with the [Cartesian coordinate system](#) in a plane. Any point can be specified by two numbers: its x and y coordinates.

```
In [ ]:  1  CartesianCoord = ...  # TODO!
         2
         3
```

- Notice the format of the data definition, especially the `(field, type)` pairs in the square brackets:

> ▸ ✅ 1. Square brackets...

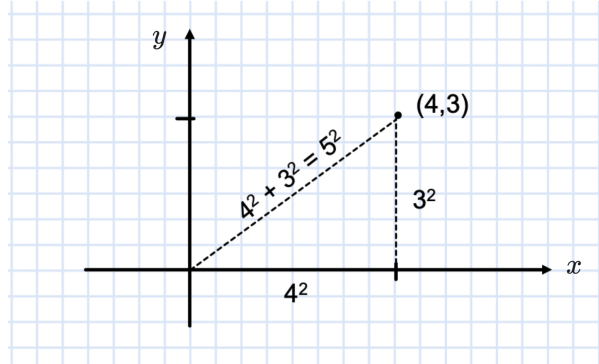> ▸ ✅ 2. Surrounding a comma-separated list...

> ▸ ✅ 3. Of (field, type) pairs

> ▸ ℹ️ Sample solution (For later. Don't peek if you want to learn 🙂)

Exercise 1: function for CartesianCoord

Problem: Design a function that takes a CartesianCoord variable and computes its distance from the origin (0,0).

Recall, the distance d from the origin can be computed using the [Pythagorean theorem](#): $d^2 = x^2 + y^2$.


```
In [ ]:   1  from math import sqrt
          2  # Hint: Pythagoras to the rescue (see picture above)
          3
          4
```

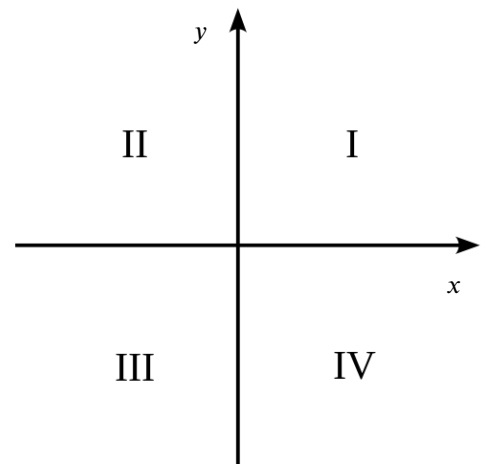> ▸ ℹ️ Sample solution (For later. Don't peek if you want to learn 🙂)

```
In [ ]:   1
          2
```

Exercise 2: another function for CartesianCoord

Problem: Design a function that takes a CartesianCoord variable and returns the corresponding quadrant.

The quadrant corresponds to the coordinates (x, y) as follows:

x	y	Quadrant
$x > 0$	$y > 0$	1
$x < 0$	$y > 0$	2
$x < 0$	$y < 0$	3
$x > 0$	$y < 0$	4

First, let's create a data definition for quadrant.

iClicker question: Which type for quadrant?

What data type should we use to represent a quadrant of the plane? Select **ALL** that apply. [Set iClicker question type to "Multiple Answer".]

A. Simple atomic:

```
        Quadrant = int
```

B. Interval:

```
        Quadrant = int  # in range [1,4]
```

C. Optional interval:

```
        Quadrant = Optional[int]  # in range [1,4]
```

D. Enumeration:

```
        Quadrant = Enum('Quadrant', [ 'I', 'II', 'III', 'IV', 'AXIS' ])
```

E. Optional enumeration:

```
        OffAxisQuadrant = Enum('OffAxisQuadrant', [ 'I', 'II', 'III', 'IV' ])
        Quadrant = Optional[OffAxisQuadrant]
```

> ▶ i Hint (for your review after class)

Quadrant data definition

In [ ]:
```
1  Quadrant = ...  # TODO!
2
3
```

> ▼ i Sample solution (For later. Don't peek if you want to learn 🙂)
>
> ```
> from typing import Optional
>
> Quadrant = Optional[int]  # in range [1,4]
> # interp. One of the 4 quadrants in the 2D Cartesian plane, or None if the coor
> dinate lies
> # along one of the axes.
>
> Q_ORIG = None
> Q1 = 1
> Q2 = 2
> Q3 = 3
> Q4 = 4
>
> @typecheck
> # Template based on Optional
> def fn_for_quadrant(q: Quadrant) -> ...:
>     if q is None:
>         return ...
>     else:
>         return ...(q)
> ```

Problem: Design a function that takes a CartesianCoord variable and returns the corresponding quadrant.

Now we can use the HtDF recipe to design our function.

```
In [ ]:  1
         2
```

iClicker question: Mixing data types

You are writing a program to track a manufacturer's line of simple toys. A toy can be made out of one material – *plastic*, *wood*, or *metal* – and it has one colour – *red*, *green*, or *blue*. Describe the best data type to represent a toy. [Set iClicker question type to "Multiple Choice".]

A. An `Enum` with two cases, each a `NamedTuple` with three attributes
B. A `NamedTuple` with two attributes, each an `Enum` with three cases
C. An `Enum` with three cases, each a `NamedTuple` with two attributes
D. A `NamedTuple` with three attributes, each an `Enum` with two cases

Exercise 3: function for two CartesianCoord variables

Problem: Design a function that takes two CartesianCoord variables and computes their distance from each other.

```
In [ ]:  1
         2
```

Exercise 4: function that returns a CartesianCoord variable

Problem: Design a function that takes two CartesianCoord variables and computes their middle point.

```
In [ ]:  1
         2
```

CartesianCoord is a simple compund with **only 2 fields(!)**, but it shows how powerful and flexible compound data can be!