In [ ]:
```
from cs103 import *
```

CPSC 103 - Systematic Program Design

Module 04 Day 2

Ian Mitchell, with thanks to Rik Blok and Giulia Toti

Make-up Monday

On Thursday, October 12, 2023 (a week from today):

- Your THURSDAY classes will be canceled. Instead, your MONDAY classes will take place at their regularly scheduled time and location on Thursday October 12.
- So **this** lecture will be canceled next Thursday. (Our lecture next Tuesday October 10 will meet as usual.)
- Thursday tutorial classes will be canceled next Thursday. (Tutorial sections next Wednesday October 11 and Friday October 13 will meet as usual.)
- Alternative plans are being developed for students in Thursday sections (T1G, T1H, T1J, T1K, T1M, and T1P **only!**). May include a Zoom session or attending another section on Wednesday October 11 or Friday October 13. Stay tuned for details.

Reminders

- this Wed-Fri: Module 4 Tutorial Attendance
- **Tue 9:30am**: Module 5: Pre-Lecture Assignment
- **Tue**: Module 4 (Compound): Worksheet
- Wed: Module 4 (Compound): Code Review
- Wed: Module 2 (HtDF): Tutorial Resubmission (optional)
- Wed: Module 4 (Compound): Tutorial Submission
- next Wed-Fri: Module 5 Week 1 Tutorial Attendance

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

---

iClicker check-in

How are you doing? Any trouble keeping up?

A. 💪 Easy-peasy... you can go faster
B. 👍 Yup, I got this
C. 😕 I might have missed a bit here or there
D. ☹️ Hmm, something's not working right
E. 😵 I have no idea what's going on

Cartesian coordinates continued

Last class we had created a compound 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 [ ]:
```
# Copied from solution constructed during last class.

from typing import NamedTuple

CartesianCoord = NamedTuple('CartesianCoord', [('x', float),
                                               ('y', float)])
# interp. coordinates of a point in the plane.
# x is the horizontal coordinate (positive to the right)
# and y is the vertical coordinate (positive upward).

CC1 = CartesianCoord(9, 3.2)
CC_ORIGIN = CartesianCoord(0,0)
CC2 = CartesianCoord(-2, -4.2)
CC_Y_AXIS = CartesianCoord(0, 99)
CC_X_AXIS = CartesianCoord(-7,0)

@typecheck
def fn_for_cartesian_coord(cc: CartesianCoord) -> ...:
    return ...(cc.x,
               cc.y)
```

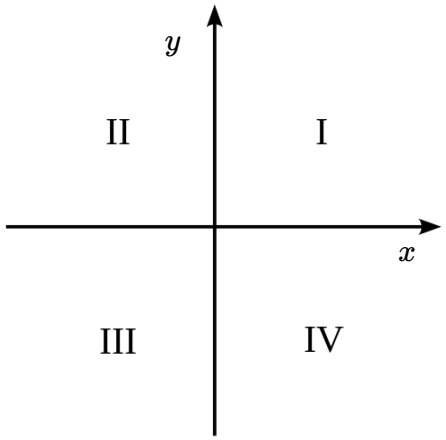
In [ ]:
```

```

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.

Quadrant data definition and a function

In [ ]:
```
# Copied from "sample solution" in last class's notebook.
# This version uses Optional wrapped around an Interval.

from typing import Optional

Quadrant = Optional[int]  # in range [1,4]
# interpr. One of the 4 quadrants in the 2D Cartesian plane, or None if the coordin
# 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 [ ]:
```
# Completed HtDF step 1 and some or all of step 2.
# TODO: Complete remaining HtDF steps.

@typecheck
def quadrant_of_coord(cc: CartesianCoord) -> Quadrant:
    """
    Return the planar quadrant associated with
    the coordinates cc.  Returns None if either
    x or y is zero.
    """
    return None # stub

start_testing()

expect(quadrant_of_coord(CC_ORIGIN), None)
expect(quadrant_of_coord(CC_X_AXIS), None)
expect(quadrant_of_coord(CC_Y_AXIS), None)
expect(quadrant_of_coord(CartesianCoord(CC1)), 1)
expect(quadrant_of_coord(CartesianCoord(CC2)), 2)
# Do we have enough tests?

summary()
```

▶ ℹ Sample solution (For later. Don't peek if you want to learn 🙂)

Function templates with more than one parameter

All the **data templates** we've constructed have a single parameter, of the data type we're designing.

But when we design a **function** that uses the data, it may have more than one parameter, of the same or different types. Which parameter do we choose to copy the data template from?

There is no firm rule. Here are some guidelines:

- If one data type is the focus of the function, choose that one.
- If one data template is more complicated, choose that one.
- Merge data templates from other parameters in.

Write a comment to indicate which template you used **and the additional parameter types you merged in**.

Example

We have completed HtDF steps 1 (*Stub*) and 2 (*Examples*). Now you complete step 3 (*Template*).

In [ ]:
```
@typecheck
def coord_is_left_of(cc: CartesianCoord, x_boundary: float) -> bool:
    """
    Returns True if CartesianCoord `cc` is to the left of
    (or has smaller x-component than) `x_boundary`.
    """
    return False # stub

start_testing()

expect(coord_is_left_of(CC0, 0), False)
expect(coord_is_left_of(CC1, 0.5), False)
expect(coord_is_left_of(CC1, 1.5), True)

summary()
```

▶ ℹ Complete code for HtDF Steps 1-3. (For later. Don't peek if you want to learn 🙂)

iClicker question

We're designing a function that takes a CartesianCoord and a Quadrant and determines if the coordinate is in the quadrant. The function's stub is shown to the right.

```python
@typecheck
def in_quad(cc:CartesianCoord,q:Quadrant)->bool:
    """
    Returns True if the Cartesian coordinate cc is in quadrant q, otherwise False.
    """
    return True # stub
```

Which would be the best template to write (after the stub)? You may assume sufficient examples have already been provided.

(A)
```python
# template from CartesianCoord with additional parameter from Quadrant
return ...(cc.x, cc.y, q)
```

(B)
```python
# template from Quadrant with additional parameter from CartesianCoord
if q == None:
    return ...(cc.x, cc.y)
else:
    return ...(q, cc.x, cc.y)
```

(C)
```python
# template from CartesianCoord and additional parameter from Quadrant
return ...(cc.x, cc.y) if q == None: return ... else: return ...(q)
```

▶ ℹ Hint (for your review after class)

Exercise 3: function for two CartesianCoord variables

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

In [ ]:

▶ ℹ Sample solution (For later. Don't peek if you want to learn 🙂)

Exercise 4: function that returns a CartesianCoord variable

A function can only return a single object. But that object can be a compound! Great way to return multiple, related pieces of information.

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

In [ ]:

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

Aside: `NamedTuple` variables are immutable

Let's try that again, this time storing the result in the fields of a new CartesianCoord variable called `middle` ...

In [ ]:
```
# Make sure you have defined the function midpoint().

# Now create a new variable and assign it the return value of a call to midpoint().

# Now project that midpoint on to the x-axis by setting its y coordinate to zero.

# Is there some other way we could create a CartesianCoord with that projected poin
```

As we see above, you can't create a `NamedTuple` and then change its field values. `NamedTuples` are *immutable* in Python (you can not change them after they are created). Good for "read-only" data where we don't want to change what we've stored. Also has performance benefits when working with lots of data.

(Python also provides the `dict` data type. It is similar to `NamedTuple` but both the fields and the field values are mutable. Mutability is a mixed blessing, so in CPSC 103 we focus on the simpler immutable `NamedTuple`)

And that concludes our CartesianCoord compound conversation 😀

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

Compounds...

- Are great for representing something complex, with multiple attributes.
- Keep related information together. Easier to keep organized than independent variables (e.g., `cc` instead of `x` and `y`).
- Let us return multiple pieces of information from a function. Remember, a function can only return one instance, but that's ok if it's a *compound*.
- Require us to provide all necessary fields to create a new instance. No chance of missing pieces of information.

iClicker check-in

How are you doing? Any trouble keeping up?

A. 💪 Easy-peasy... you can go faster
B. 👍 Yup, I got this
C. 😕 I might have missed a bit here or there
D. ☹️ Hmm, something's not working right
E. 😵 I have no idea what's going on

Artist: Compound, enumeration, or simple atomic?

Our artist question asks you to represent "an artist's family name, given name, birthplace, and art form (e.g., oil painting, sculpture, dance)".

What does one value of this type look like? Let's use Georgia O'Keeffe (the painter, born in Wisconsin) as our example. We'll try solving the problem first with a compound and then with an enumeration and then simple atomic data to try to represent O'Keeffe.

Version 1: Artist data definition as a compound

In [ ]:
```
from typing import NamedTuple

Artist = NamedTuple('Artist', [('family_name', str),
                               ('given_name', str),
                               ('birthplace', str),
                               ('art_form', str)])
# interp. an artist with their family name, given name, place of birth,
#         and the art form they were best known for.
A_MONET = Artist('Monet', 'Claude', 'Paris', 'pastels')
A_VAN_GOGH = Artist('van Gogh', 'Vincent', 'Netherlands', 'oil paintings')

@typecheck
# template based on compound (4 fields)
def fn_for_artist(a: Artist) -> ...:
    return ...(a.family_name,
               a.given_name,
               a.birthplace,
               a.art_form)
```

How do we represent Georgia O'Keeffe with a compound?

In [ ]:
```
# Georgia O'Keefe was born in Wisconsin and is known for her watercolors
```

▶ ℹ Sample solution (For later. Don't peek if you want to learn 🙂)

Version 2: Artist data definition as an enumeration

In [ ]:
```
from enum import Enum

Artist = Enum('Artist', ['family_name', 'given_name', 'birthplace', 'art_form'])
# interp. an aspect of an artist, one of their family name, their given name
# their birthplace, or their art form.
# Examples are redundant for enumerations.


# template based on enumeration (4 cases)
@typecheck
def fn_for_artist(a: Artist) -> ...:
    if a == Artist.family_name:
        return ...
    elif a == Artist.given_name:
        return ...
    elif a == Artist.birthplace:
        return ...
    elif a == Artist.art_form:
        return ...
```

How do we represent Georgia O'Keeffe with an enumeration?

In [ ]:
```
# Georgia O'Keefe was born in Wisconsin and is known for her watercolors
```

▶ ℹ Sample solution (For later. Don't peek if you want to learn 🙂)

Version 3: Artist data definition as a simple atomic

In [ ]:
```
Artist = str
# interp. an artist with their family name then given name followed by "born in"
# and their birthplace and "known for" and their art form.
A_MONET = 'Monet Claude born in Paris known for pastels'
A_VAN_GOGH = 'van Gogh Vincent born in Netherlands known for oil paintings'

@typecheck
# template based on atomic non-distinct
def fn_for_artist(a: Artist) -> ...:
    return ...(a)

# How do we represent Georgia O'Keeffe?
```

How do we represent Georgia O'Keeffe with simple atomic data?

In [ ]:
```
# Georgia O'Keefe was born in Wisconsin and is known for her watercolors
```

▶ ℹ Sample solution (For later. Don't peek if you want to learn 🙂)

Freestyle 🙂

Think of something you're interested in, something meaningful to you. Music? Movies? Books? Sports? Burritos?

How could you design a data definition to represent information about that domain in a compound?

In the cell below, start designing your data with the HtDD recipe!

You should probably limit yourself to four or five fields to keep the task manageable. Maybe one of those fields will be an enumeration or an optional.

In [ ]:
```
# freestyle!
```