Outline

1. **Histograms**
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. **Arguments of Functions**
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions

MCS 260 Lecture 14
Introduction to Computer Science
Jan Verschelde, 28 June 2023
Histograms
interpreting results of a simulation

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:
1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.
Histograms with `matplotlib`

On data randomly generated from a normal distribution, with `matplotlib` (requires `numpy`) we can plot:
global & local variables
arguments of functions

1. Histograms
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. Arguments of Functions
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):
1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.
Flowchart of the voting machine

\[
(t_{\text{yes}}, t_{\text{no}}) = (0, 0)
\]

\[
v = \text{input}('\text{approve ? (y/n)}')
\]

\[
v == '0'?
\]

\[
\text{False}\quad \text{print}(t_{\text{yes}}, t_{\text{no}})
\]

\[
v == 'y'?
\]

\[
\text{False}\quad t_{\text{no}} = t_{\text{no}} + 1
\]

\[
\text{True}\quad t_{\text{yes}} = t_{\text{yes}} + 1
\]
global & local variables
arguments of functions

1. Histograms
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. Arguments of Functions
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
global and local variables
hierarchy imposed on data

In top down design we distinguish between

- functions that focus on one particular task
- the main program that calls the functions

Also the data fits into two categories:

- variables inside a function are \textit{local}
- data managed by the main program is \textit{global}

Example, in the voting machine:

- the variable to store the answer will be local
- the tally of the votes is global
Python functions are functions

A function $f$ in the proper mathematical sense, called like $y = f(x)$, does not change the argument $x$ of the function.

Updating the tally $t$ with vote $v$ with the function $\text{update}(t, v)$, called as $t = \text{update}(t, v)$, where $t = (\text{tyes}, \text{tno})$.

The function $\text{update}$ will

1. check the value of the vote $v$
2. create a new tuple with updated values
3. return the new tuple

The caller of $\text{update}(t, v)$ assigns the updated values to $\text{tyes}$ and $\text{tno}$.

Some terminology:

- call by value: with tuples (immutable)
- call by reference: with lists (variable).
global & local variables
arguments of functions

1. Histograms
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. Arguments of Functions
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
Call by Value works via *tuple assignment*.

\[
T = (3, 4) \\
vote = 1 \quad \# \text{we have a yes vote} \\
R = (T[0] + 1, T[1])
\]

The \( T = \text{update}(T, \text{vote}) \) after the call to update is a tuple assignment.

Call by Reference works via *item assignment* in a list.

\[
L = [3, 4] \\
vote = 0 \quad \# \text{we have a no vote} \\
L[1] = L[1] + 1
\]

The \( \text{update}(L, \text{vote}) \) does an item assignment in the list \( L \).
global & local variables
arguments of functions

1. Histograms
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. Arguments of Functions
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
Arguments of Variable Length

Consider the area computation of a square or rectangle. The dimensions of a rectangle are length and width, but for a square we only need the length. → functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the tuple must
- appear after all other arguments `args`,
- and be preceded by `*`. 
global & local variables
arguments of functions

1. Histograms
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. Arguments of Functions
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
If arguments are optional, then we may identify the extra arguments of a function with keywords.

Instead of \( a = \text{area}(L,W) \)
we require \( a = \text{area}(L, \text{width}=W) \).

Python syntax:

```python
def < f > ( < a > , * < t > , ** < dict > ) :
```

The name of the dictionary `dict` must
- appear at the very end of the arguments,
- and be preceded by `**`. 

```python
local variables and arguments
```
1. **Histograms**
   - tallying the votes
   - global and local variables
   - call by value or call by reference

2. **Arguments of Functions**
   - arguments of variable length
   - using keywords for optional arguments
   - functions using functions
functions using functions

To approximate the integral of a function $f(x)$ over $[a, b]$, the trapezoidal rule is

$$\int_a^b f(x)\,dx \approx \frac{1}{2}(f(a) + f(b))(b - a).$$

Geometrically, we approximate the area under $f(x)$ for $x \in [a, b]$ by the area of a trapezium, with base $[a, b]$ and heights $f(a)$ and $f(b)$.

A function as argument of a Python function, template:

```python
def <rule>(<f>, <a>, <b>):
    "integrate function f over [a,b]"
```
Exercises

1. Simulate a coin toss in a Python program, applying `random.randint(0, 1)` at least a thousand times. Count the number of 0s and 1s. Is Python’s coin fair?

2. Make a histogram of the normal distribution, using `gauss` of the `random` module. Generate 10,000 numbers using 200 as mean and 25 as standard deviation. Distribute the generated numbers in 50 buckets of equal width between 0 and 300. The first bucket counts the numbers in \([0, 6)\), the second in \([6, 12)\), etc.

3. Modify the vote tally to include abstain votes.

4. Define a function to compute the volume of a cube, or general parallelepiped. For a cube, only one parameter will be given, otherwise, the user must specify length, width, and height.