Welcome to COMP1081 - Algorithms and Data Structures

The lecture will begin at 5.05 p.m. While you wait ...

We will use Poll Everywhere during the lecture for you to answer questions (anonymously).

You can either download the Poll Everywhere app and join the presentation `eamonn` or respond on the web at `https://pollev.com/eamonn`

You do not need to sign up or log in. A first question should appear soon.

(Don't worry if you can't access it. I will display the questions on the screen during the lecture.)

COMP1081 - Algorithms and Data Structures Part 1

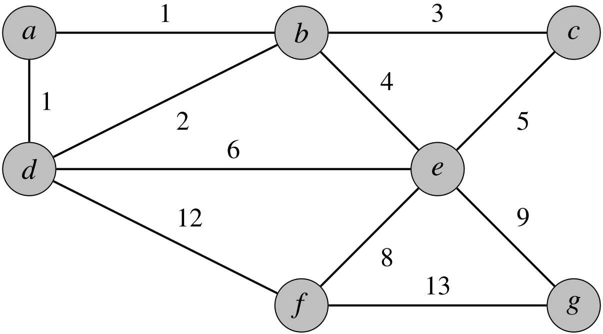
Topic 1: Introduction and Pseudocode

Eamonn Bell

eamonn.bell@durham.ac.uk

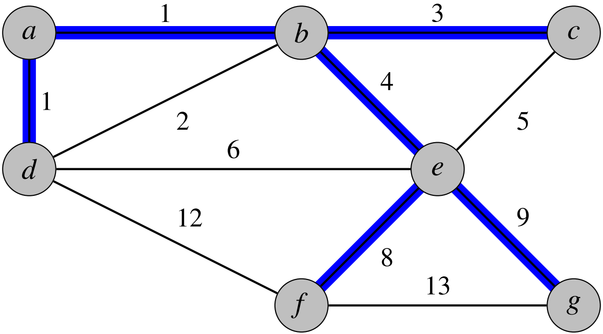
(Slides based on MJ, SD, and other former module staff)

Algorithms and Data Structures?



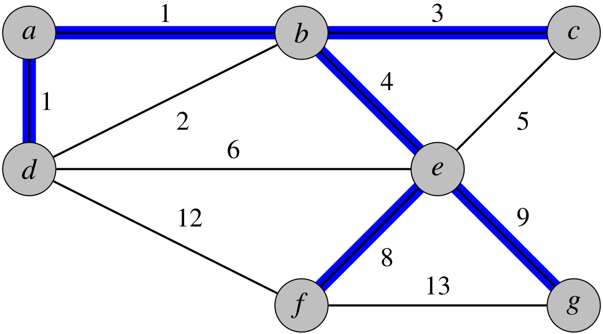
Links in a network have costs. How can we connect all the nodes as cheaply as possible.

Algorithms and Data Structures?



Here is a solution.

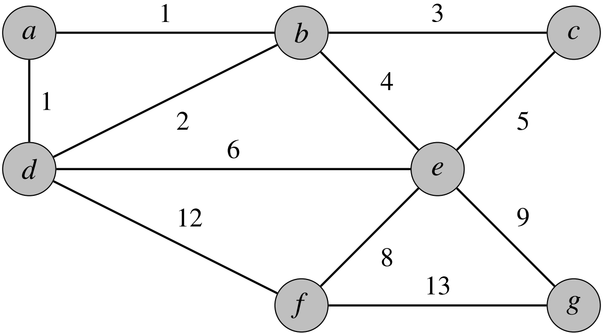
Algorithms and Data Structures?



Here is a solution.

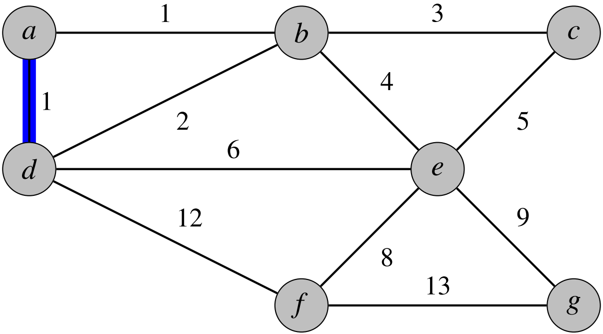
But can we describe how we found it — so that we would know what to do if presented with a similar problem with millions of nodes

Algorithms and Data Structures?



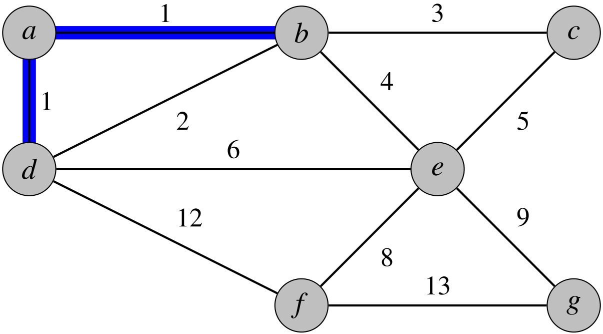
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



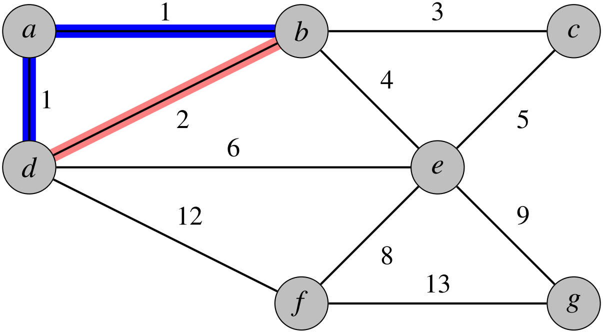
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



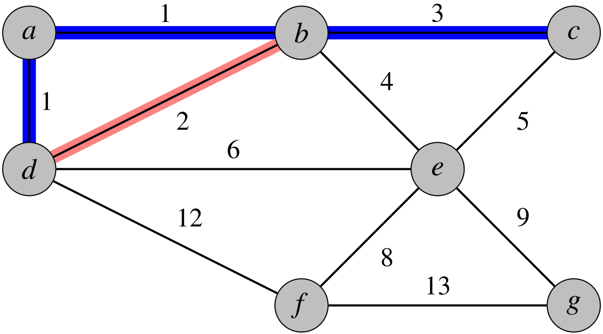
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



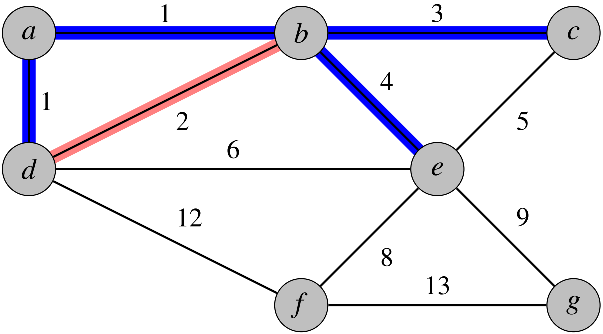
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



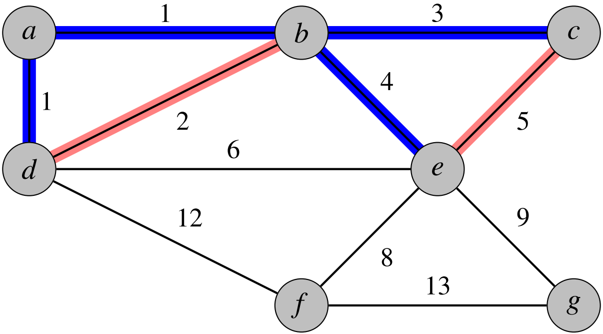
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



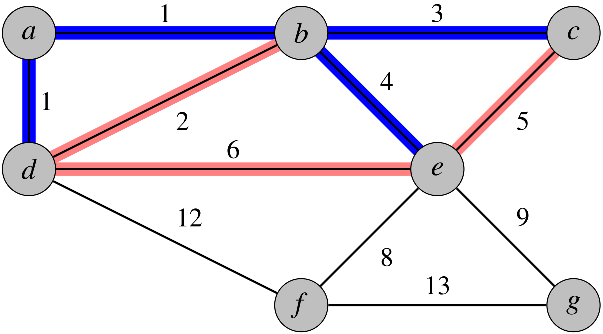
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



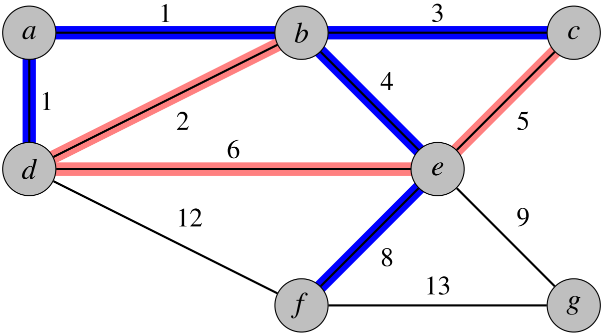
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



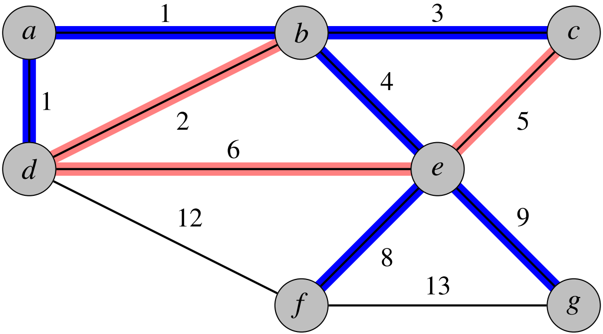
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



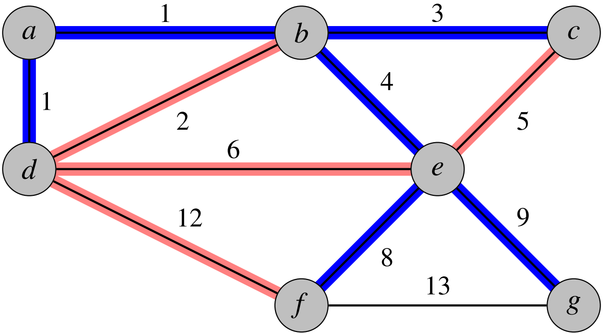
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



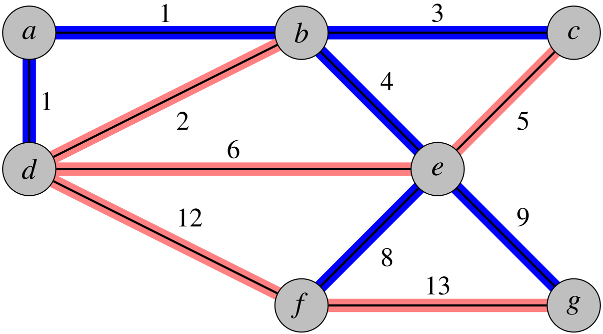
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



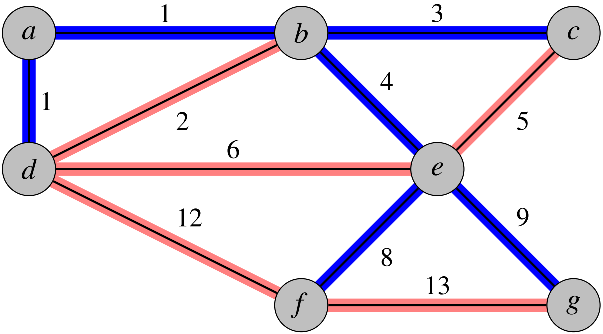
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



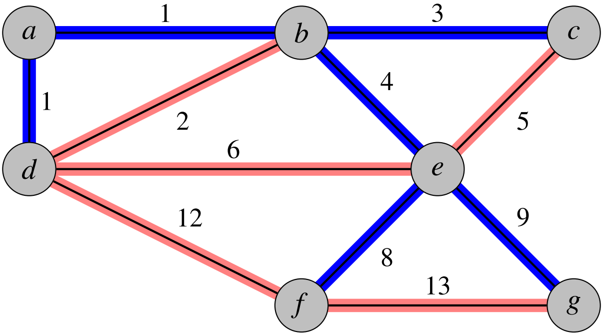
Repeatedly pick the cheapest edge that makes new connections and add to the solution.

Algorithms and Data Structures?



We have described an algorithm to solve the problem. But are we sure it is correct? Is it practical?

Algorithms and Data Structures?



If we wanted to solve this program on a computer, how would we store the network (the data). Would this affect the efficiency of our algorithm?

Algorithms

What is an “algorithm”?

Algorithms

What is an “algorithm”?

> *An algorithm is a method or a process followed to solve a problem.*

What properties must an algorithm have?

Algorithms

What is an “algorithm”?

An algorithm is a method or a process followed to solve a problem.

What properties must an algorithm have?

- Correctness.
- Composed of concrete unambiguous steps.
- The number of steps must be finite.
- Must terminate.

Data Structures

What is a “data structure”?

Data Structures

What is a “data structure”?

> *A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.*

Data Structures

What is a "data structure"?

> *A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.*

Why do we study data structures and algorithms?

Data Structures

What is a "data structure"?

A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Why do we study data structures and algorithms?

We want to solve problems efficiently – to make the best use of resources such as space and time. Our choice of data structure or algorithm can make the difference between a program running in a few seconds or many days.

In this module

- Learn the commonly used data structures and when to use them. These form a programmer's basic data structure "toolkit."
- Study well-known algorithmic techniques and demonstrate their application.
- Understand how to measure the cost of a data structure or an algorithm. These techniques also allow you to judge the merits of new data structures and algorithms that you or others might invent.

Module Content

- Introduction & pseudocode
- Basic data structures
- Recursive algorithms
- Analysing algorithms (asymptotic classes)
- Sorting
- Binary search
- Graph algorithms

Module Information

- 40 one-hour lectures. 19 one-hour practicals.

Module Information

- 40 one-hour lectures. 19 one-hour practicals.
- Lecturers: Eamonn Bell, Thomas Erlebach (module coordinator), Anish Jindal and Amitabh Trehan

Module Information

- 40 one-hour lectures. 19 one-hour practicals.
- Lecturers: Eamonn Bell, Thomas Erlebach (module coordinator), Anish Jindal and Amitabh Trehan
- Course text: Data Structures and Algorithms in Python (or Java) by Goodrich et al.

Module Information

- 40 one-hour lectures. 19 one-hour practicals.
- Lecturers: Eamonn Bell, Thomas Erlebach (module coordinator), Anish Jindal and Amitabh Trehan
- Course text: Data Structures and Algorithms in Python (or Java) by Goodrich et al.
- Course materials are on `Learn Ultra`

Module Information

- 40 one-hour lectures. 19 one-hour practicals.
- Lecturers: Eamonn Bell, Thomas Erlebach (module coordinator), Anish Jindal and Amitabh Trehan
- Course text: Data Structures and Algorithms in Python (or Java) by Goodrich et al.
- Course materials are on `Learn Ultra`
- Assessment:
  - Assignment due 23rd January 2025. (More info on Sharepoint > Department of Computer Science Undergraduate Community)
  - End of year exam.

Module Information

- 40 one-hour lectures. 19 one-hour practicals.
- Lecturers: Eamonn Bell, Thomas Erlebach (module coordinator), Anish Jindal and Amitabh Trehan
- Course text: Data Structures and Algorithms in Python (or Java) by Goodrich et al.
- Course materials are on `Learn Ultra`
- Assessment:
  - Assignment due 23rd January 2025. (More info on Sharepoint > Department of Computer Science Undergraduate Community)
  - End of year exam.
- EB Office hours: in person at MCS 2105 10 a.m. – 11 a.m. on Thursdays during Part I, email for appointment otherwise.

Module Activities

- This module is one of six modules you have this year . . .
- . . . so might be expected to consume about six hours of your time each week (assuming a full-time work pattern).
- Three hours on lectures — attending (2 hours), reviewing (1 hour)
- Three hours on practicals — preparing, attending (2 hours), reviewing

Practicals

Practicals begin on the week starting 14th October 2025. Attendance is taken. Practicals are an essential opportunity to put knowledge from lectures into practice through writing pseudocode, implementing (in Python), and working on problem sets.

- Monday 11 a.m. @ MCS3097 (Jiangtao and Blair)
- Monday 11 a.m. @ MCS2094 (Ben and Josh)
- Tuesday 11 a.m. @ MCS3097 (Amren and Yoyo)
- Tuesday 11 a.m. @ MCS3098 (Jingtao and Karmen)
- Tuesday 4 p.m. (to be confirmed)
- Thursday 9 a.m. (Ben and Jude)
- Friday 11 a.m. (Ben and Carlin)
- Friday 2 p.m. (Carlin and Jude)

ADS – Module Evaluation Questionnaire (MEQ)

- Students fill in MEQs for ADS each year around the end of the 2nd term

- **Examples of issues raised in last year's MEQs:**
  - Students with lack of programming experience **struggled with the programming part** of the coursework.
  - The coursework involved **OOP** in Python, but OOP is only taught in Programming Paradigms in Y2.
  - Part 2 only had one main Powerpoint which sometimes made it **difficult to locate certain topics** when revisiting.
  - Part 4 (Graph theory) was **too rushed**.

Examples of changes made in 24/25 in response to the MEQ feedback

- Additional guidance for Python implementation added in Part 1 practicals
- Ensured that only the programming concepts covered in CT are required to solve the coursework
- Slides for Part 2 now also provided for each topic covered in Part 2 separately
- Content and delivery of Part 4 revised and streamlined

Durham University

RAM model of computation

This is a highly simplified model of computation that we use compute the efficiency of an algorithm.

Random access machine:

1. Memory consists of an infinite array.
2. Instructions are executed sequentially one at a time.
3. All instructions take unit time. Running time is the number of instructions executed.

Pseudocode

Algorithm to connect network cheaply

```
Input: a network with costs for links
Output: a least cost "tree" that connects the network
  $V$ is the set of nodes
  $E$ is the set of links
  $A$ is the empty set
  sort $E$ so that links are in order of increasing cost
  while $E$ is not empty do
      choose $e$ in $E$ with minimum cost
      if $A + e$ contains no cycle then
          add $e$ to $A$
      end if
      remove $e$ from $E$
  end while
  return $A$
```

Pseudocode

To describe algorithms we will use generic pseudocode, not any one programming language.

No (very) strict rules

Typical “framework”:

```
Algorithm: Foo
Input: numbers $a_1, a_2, \ldots, a_n$
Output: $\max\{a_i | 1 \leq i \leq n\}$
    PSEUDOCODE that solves the problem goes here
```

Pseudocode: variables

Will (often) need to use variables
Basic types: integer, float, char(acter), string

If you've got an "important" variable then explicitly declare it, i.e., say what type it will be storing

```
integer i
float f
char c
string s
```

Pseudocode: variables

Of course, you will want to assign values to variables
No real convention here; some (real) languages use “=”, some use “:=”, and others use “←”

```
integer i = 0
string s := "test string"
char c ← ‘a’
```

(Many languages use double quotes for strings, and single quotes for characters)

Pseudocode: variables

May also have declaration separate from initialisation, e.g.

```
integer i
i = 12
```

Also, may have multiple variables in one go:

```
integer x=0, y=2, z=3
```

Arithmetic operations

- **(. . .)** parentheses (or brackets) for grouping
- **+ - * /** add, subtract, multiply, divide

Examples

```
volume = length * width * height
z = (x+1) * y / (a-b)
```

Logical operations: AND, OR, NOT

Output (Printing)

Keyword “print”

Examples

```
print x
print "Hello world"
print "value of z is ", z
```

may produce outputs

```
0
Hello world
value of z is 12
```

Notice comma for concatenation (elsewhere may see ‘+’)

If-then-else

Simple if-then

```
if condition then
    statement
    statement
end if
```

Many conditions involve comparisons:

`<`   `>`   `<=` ($\leq$)   `>=` ($\geq$)   `==` ($=$)   `!=` ($\neq$)

If-then-else

Simple if-then

```
if condition then
    statement
    statement
end if
```

Many conditions involve comparisons:

$<$ $>$ <= (≤)   >= (≥)   == (=)   != (≠)

```
if $x \neq 0$ then
    $z = y/x$
end if
```

It's good pseudocode style to visually indent the "body" of a conditional statement

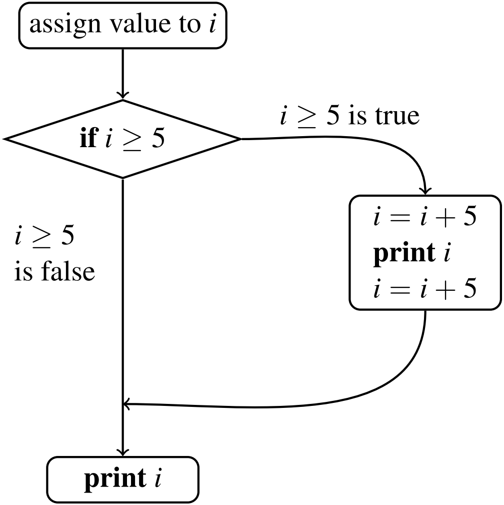
If-then-else: Question 1

```
$i =$
if $i \geq 5$ then
    $i = i + 5$
    print $i$
end if
```

If-then-else: Question 2

```
$i =$
if $i \geq 5$ then
    $i = i + 5$
    print $i$
    $i = i + 5$
end if
print $i$
```

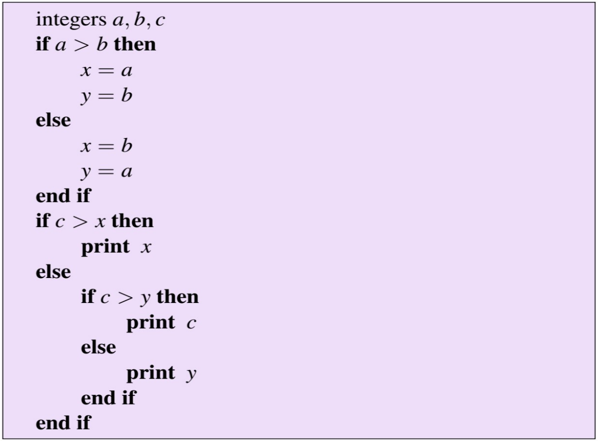
If-then flow diagram for Question 2



```
$i =$
**if** $i \geq 5$ **then**
    $i = i + 5$
    **print** $i$
    $i = i + 5$
**end if**
**print** $i$
```

Welcome to Algorithms and Data Structures

The lecture will begin at 11.05am. While you wait, consider this pseudocode and work out what it prints (this is Question 3 below).



```
integers $a, b, c$
if $a > b$ then
    $x = a$
    $y = b$
else
    $x = b$
    $y = a$
end if
if $c > x$ then
    print $x$
else
    if $c > y$ then
        print $c$
    else
        print $y$
    end if
end if
```

If-then-else

What if we want to do something when the condition is false? Could, of course, write

```
if $x \neq 0$ then
    $z = y/x$
end if
if $x = 0$ then
    print "division by zero error"
end if
```

If-then-else

What if we want to do something when the condition is false? Could, of course, write

```
if $x \neq 0$ then
    $z = y/x$
end if
if $x = 0$ then
    print “division by zero error”
end if
```

However, this is nicer:

```
if $x \neq 0$ then
    $z = y/x$
else
    print “division by zero error”
end if
```

Nested conditionals

Can of course construct more complicated things:

```
if condition then
    statement
    if condition then
        statement
        statement
    else
        statement
    end if
else
    if condition then
        statement
    end if
end if
```

Nested conditionals

But care needed! Or: one reason for explicit "end if".

```
$n = 0$
integers $a$, $b$ and $c$
**if** $a > b$ **then**
**if** $a > c$ **then**
$n = 1$
**else**
$n = 2$
**print** $n$
```

Nested conditionals

But care needed! Or: one reason for explicit "end if".

```
$n = 0$
integers $a$, $b$ and $c$
**if** $a > b$ **then**
    **if** $a > c$ **then**
        $n = 1$
    **else**
        $n = 2$
    **end if**
**end if**
**print** $n$
```

Nested conditionals

But care needed! Or: one reason for explicit "end if".

```
$n = 0$
integers $a$, $b$ and $c$
**if** $a > b$ **then**
    **if** $a > c$ **then**
        $n = 1$
    **end if**
**else**
    $n = 2$
**end if**
**print** $n$
```

If-then-else: Question 3

```
integers $a, b, c$
if $a > b$ then
    $x = a$
    $y = b$
else
    $x = b$
    $y = a$
end if
if $c > x$ then
    print $x$
else
    if $c > y$ then
        print $c$
    else
        print $y$
    end if
end if
```

For loop

If you want to iterate some (numerical) variable through some range

Great many variations in how languages do this, simplest is probably

```
for variable = lower to upper do
    body (will often depend on variable)
end for
```

“Body” is simply a sequence of statements (and may contain If-Then-Elses, other loops, whatever)

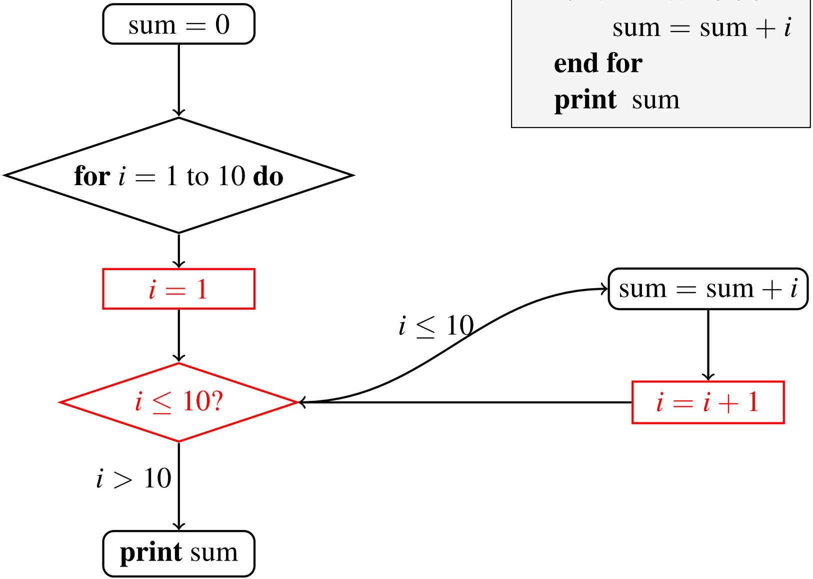
For loop example

.

```
integer sum $=0$
**for** $i = 1$ to $10$ **do**
    sum $=$ sum $+\, i$
**end for**
**print** sum
```

For loop flow diagram

```
integer sum $=0$
**for** $i = 1$ to $10$ **do**
    sum $=$ sum $+ i$
**end for**
**print** sum
```



You'll have noticed that the previous **for** loop can only iterate consecutive integers.

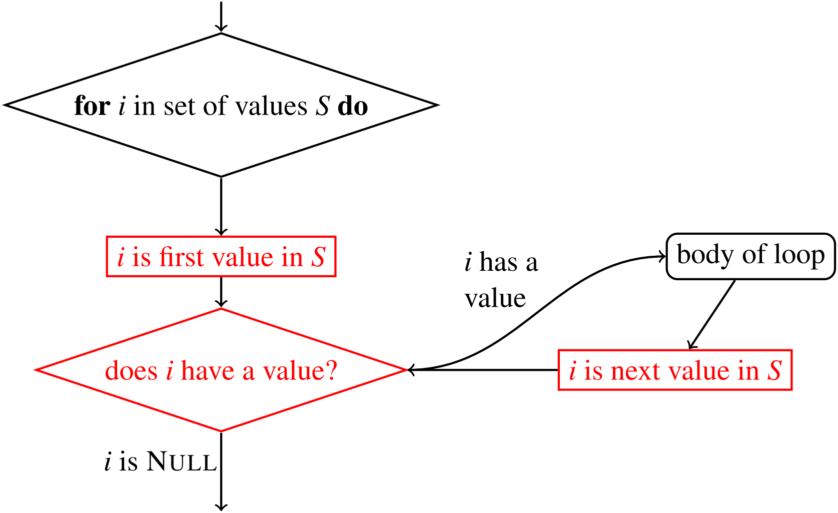
There's another, more generic one: iterate over given base set

```
for value in {value1, value2, ...} do
    body (will often depend on value)
end for
```

Example

```
integer sum = 0
for prime in { 2, 3, 5, 7 } do
    sum = sum+ prime
end for
```

generic for loop flow diagram



for loops: Question 4

Or we can iterate some variable through a range, but increment by a stated value.

```
for $i = 0$ to 9; $i += 2$ do
    print $i$
end for
```

for loops: Question 5

```
for $i = 0$ to 9; $i \mathrel{+}= -1$ do
    print $i$
end for
```

for loops: Question 5

Clearer to write:

```
for $i = 9$ to 0; $i \mathrel{+}= -1$ do
    print $i$
end for
```

While loop

Do something while some condition is true

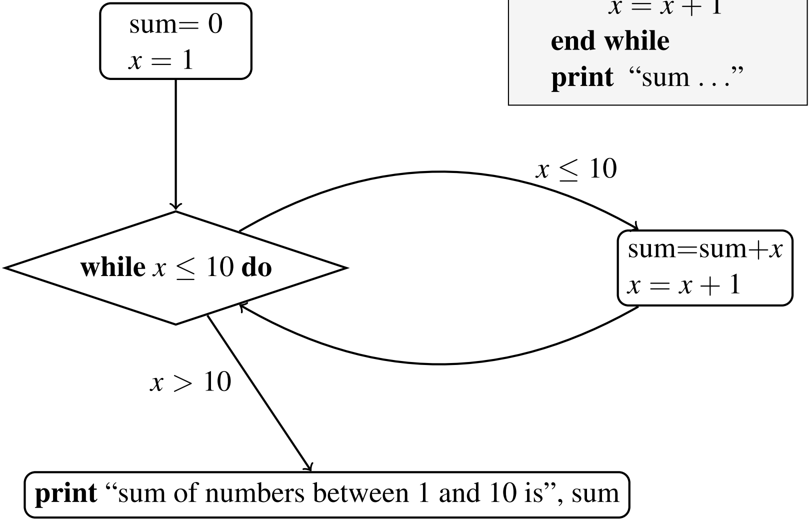
```
while condition do
    body
end while
```

E.g.

```
sum $=0$
$x=1$
while $x \leq 10$ do
    sum $=$ sum $+x$
    $x=x+1$
end while
print "sum of numbers between 1 and 10 is", sum
```

While loop flow diagram

```
sum = 0
$x = 1$
**while** $x \leq 10$ **do**
    sum = sum $+x$
    $x = x + 1$
**end while**
**print** “sum . . .”
```


sum= 0
$x = 1$
while $x \leq 10$ **do**
$x \leq 10$
sum=sum$+x$
$x = x + 1$
$x > 10$
print “sum of numbers between 1 and 10 is”, sum

Important difference between **for** and **while** over numerical values:
for increments loop-variable automatically; **while** doesn't

```
for $i = 1$ to 10 do
    print $i$
end for
```

```
$i = 1$
while $i \leq 10$ do
    print $i$
    $\mathbf{i} = \mathbf{i} + \mathbf{1}$        $\longleftarrow$ need to increment $i$ by hand
end while
```

while loops: Question 6

```
product $= 1$
$x = 0$
while $x \leq 5$ do
    product $=$ product $\times x$
end while
print product
```

while loops: Question 7

.

```
positive integer $a$
positive integer $b$
integer $s$ = 0
integer count = 1
while count $\leq a$ do
    $s = s + b$
    count $=$ count $+1$
end while
print $s$
```

while loops: Question 8

.

```
positive integer $n$
integers $a_1, a_2, a_3, \ldots, a_n$
integer $s = 0$
for $i = 1$ to $n$ do
    $s = s + a_i$
end for
$s = s/n$
print $s$
```

Everything can be nested: Question 9

```
for $x = 1$ to 4 do
    for $y = 1$ to 4 do
        go to coordinate $(x, y)$
        plot red circle of diameter $0.1$
    end for
end for
```

Everything can be nested: Question 9

```
for $x = 1$ to $4$ do
    for $y = 1$ to $4$ do
        go to coordinate $(x, y)$
        plot red circle of diameter $0.1$
    end for
end for
```

What if the first two lines were swapped?