COMP110: Principles of Computing

10: References

FALMOUTH UNIVERSITY

Research journal

Research journal

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them
- **Research** how this paper has influenced the field of computing

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them
- **Research** how this paper has influenced the field of computing
- **Write up** your findings

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them
- **Research** how this paper has influenced the field of computing
- **Write up** your findings
  - Maximum 1500 words

Research journal

- **Read** some seminal papers in computing (listed on the assignment brief)
- **Choose** one of them
- **Research** how this paper has influenced the field of computing
- **Write up** your findings
  - Maximum 1500 words
  - With reference to appropriate academic sources

Marking rubric

See assignment brief on LearningSpace/GitHub

Timeline

- **Peer review** next week! (4th December)
- **Deadline** shortly after! (check MyFalmouth)

FALMOUTH UNIVERSITY

Pass by reference

References

References

- Our picture of a variable: a labelled box containing a value

References

- Our picture of a variable: a labelled box containing a value
- For “plain old data” (e.g. numbers), this is accurate

References

- Our picture of a variable: a labelled box containing a value
- For "plain old data" (e.g. numbers), this is accurate
- For **objects** (i.e. instances of classes), variables actually hold **references** (a.k.a. **pointers**)

References

- Our picture of a variable: a labelled box containing a value
- For "plain old data" (e.g. numbers), this is accurate
- For **objects** (i.e. instances of classes), variables actually hold **references** (a.k.a. **pointers**)
- It is possible (indeed common) to have **multiple references** to the same underlying object

The wrong picture

```python
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

The wrong picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

Variable	Value
x	
y	
z	

The wrong picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

<table>
<tr><th>Variable</th><th>Value</th></tr>
<tr><td>x</td><td><table><tr><td>a</td><td>30</td></tr><tr><td>b</td><td>40</td></tr></table></td></tr>
<tr><td>y</td><td></td></tr>
<tr><td>z</td><td></td></tr>
</table>

The wrong picture

```python
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

Variable	Value	
x	a	30
	b	40
y	a	50
	b	60
z		

The wrong picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

Variable	Value	
x	a	30
	b	40
y	a	50
	b	60
z	a	50
	b	60

The right picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

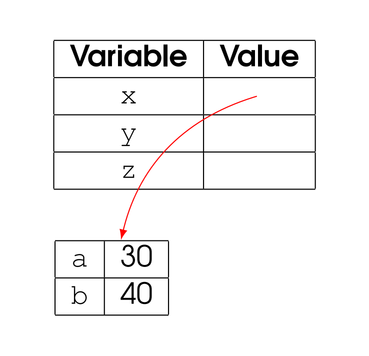
The right picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```

Variable	Value
x	
y	
z	

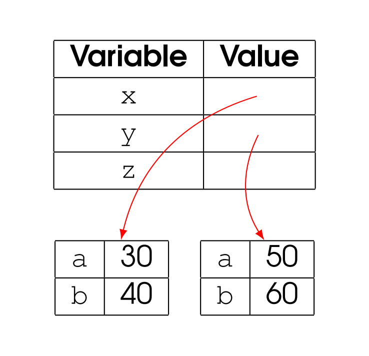
The right picture

```python
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```



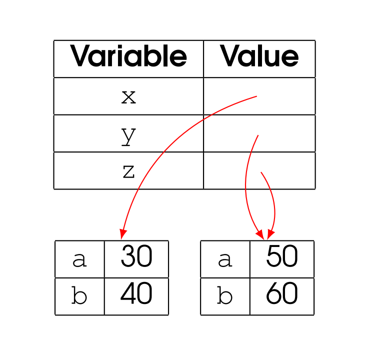
The right picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```



The right picture

```
class Thing:
    def __init__(self,
                 a, b):
        self.a = a
        self.b = b

x = Thing(30, 40)
y = Thing(50, 60)
z = y
```



Values and references

Socrative room code: `FALCOMPED`

```
a = 10
b = a
a = 20
print("a:", a)
print("b:", b)
```

Values and references

Socrative room code: `FALCOMPED`

```
class X:
    def __init__(self, value):
        self.value = value

a = X(10)
b = a
a.value = 20
print("a:", a.value)
print("b:", b.value)
```

Values and references

Socrative room code: `FALCOMPED`

```
class X:
    def __init__(self, value):
        self.value = value

a = X(10)
b = X(10)
a.value = 20
print("a:", a.value)
print("b:", b.value)
```

Pass by value

Pass by value

In **function parameters**, “plain old data” is passed by **value**

Pass by value

In **function parameters**, “plain old data” is passed by **value**

```
def double(x):
    x *= 2

a = 7
double(a)
print(a)
```

Pass by value

In **function parameters**, “plain old data” is passed by **value**

```
def double(x):
    x *= 2

a = 7
double(a)
print(a)
```

`double` does not actually do anything, as `x` is just a local copy of whatever is passed in!

Pass by reference

Pass by reference

However, instances are passed by **reference**

```
class Box:
    def __init__(self, v):
        self.value = v

def double(x):
    x.value *= 2

a = Box(7)
double(a)
print(a.value)
```

Pass by reference

However, instances are passed by **reference**

```
class Box:
    def __init__(self, v):
        self.value = v

def double(x):
    x.value *= 2

a = Box(7)
double(a)
print(a.value)
```

`double` now has an effect, as `x` gets a reference to the `Box` instance

Lists are objects too

Lists are objects too

```
a = ["Hello"]
b = a
b.append("world")
print(a)   # ["Hello", "world"]
```

Lists are objects too

```
a = ["Hello"]
b = a
b.append("world")
print(a)   # ["Hello", "world"]
```

... which means you should be careful when passing lists into functions, because the function might actually change the list!

References can be circular

```python
class X:
    pass

foo = X()
foo.x = foo
foo.y = "Hello"

print(foo.x.x.x.x.x.y)
```

References and pointers

References and pointers

- Some languages (e.g. C, C++) use **pointers**

References and pointers

- Some languages (e.g. C, C++) use **pointers**
- Pointers are a type of reference, and have the same semantics

References and pointers

- Some languages (e.g. C, C++) use **pointers**
- Pointers are a type of reference, and have the same semantics
- C++ also has something called references...

Vectors

2D vectors

2D vectors

- A **2D vector** is represented by a **pair** of **numbers**

2D vectors

- A **2D vector** is represented by a **pair** of **numbers**
- Often represented as a **column vector**

2D vectors

- A **2D vector** is represented by a **pair** of **numbers**
- Often represented as a **column vector**
- E.g. $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ -4 \end{pmatrix}$ or $\begin{pmatrix} -3.7 \\ 6.2 \end{pmatrix}$

2D vectors

- A **2D vector** is represented by a **pair** of **numbers**
- Often represented as a **column vector**
- E.g. $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ -4 \end{pmatrix}$ or $\begin{pmatrix} -3.7 \\ 6.2 \end{pmatrix}$
- General form: $\begin{pmatrix} x \\ y \end{pmatrix}$

2D vectors

- A **2D vector** is represented by a **pair** of **numbers**
- Often represented as a **column vector**
- E.g. $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ -4 \end{pmatrix}$ or $\begin{pmatrix} -3.7 \\ 6.2 \end{pmatrix}$
- General form: $\begin{pmatrix} x \\ y \end{pmatrix}$
- Can also have $3, 4, 5, \ldots$ dimensional vectors

Vectors as points

Vectors as points

- $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the **origin**

Vectors as points

- $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the **origin**
- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents a point x units to the right and y units up from the origin

Vectors as points

- $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the **origin**
- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents a point x units to the right and y units up from the origin
  - Negative values represent left and down

Vectors as points

- $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the **origin**
- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents a point x units to the right and y units up from the origin
  - Negative values represent left and down
  - In computer graphics, sometimes y points down instead of up

Operations on vectors

Operations on vectors

- Addition and subtraction work **element-wise**

Operations on vectors

- Addition and subtraction work **element-wise**
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$

Operations on vectors

- Addition and subtraction work **element-wise**
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix}$

Operations on vectors

- Addition and subtraction work **element-wise**
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix}$
- Multiplication by a **scalar** (a number) also works element-wise

Operations on vectors

- Addition and subtraction work **element-wise**
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}$
  - $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix}$
- Multiplication by a **scalar** (a number) also works element-wise
  - $c \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c \times x \\ c \times y \end{pmatrix}$

Vectors as offsets

Vectors as offsets

- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents an offset of x units to the right and y units up

Vectors as offsets

- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents an offset of x units to the right and y units up
- Subtraction: if p and q are points, then $q - p$ is the offset of q relative to p

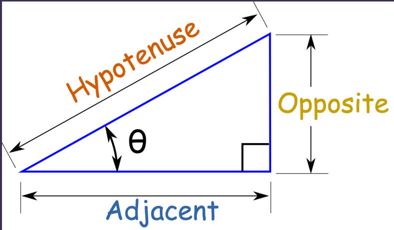
Vectors as offsets

- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents an offset of x units to the right and y units up
- Subtraction: if p and q are points, then $q - p$ is the offset of q relative to p
- Addition: if p is a point and u is an offset, then $p + u$ is the point at an offset of u from p

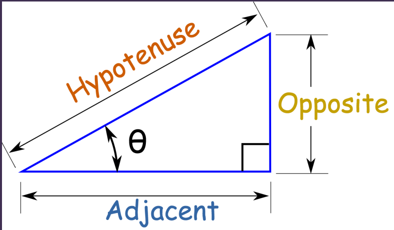
Vectors as offsets

- $\begin{pmatrix} x \\ y \end{pmatrix}$ represents an offset of x units to the right and y units up
- Subtraction: if p and q are points, then $q - p$ is the offset of q relative to p
- Addition: if p is a point and u is an offset, then $p + u$ is the point at an offset of u from p
- Addition: if u and v are offsets, then $u + v$ is the combined offset

Trigonometry

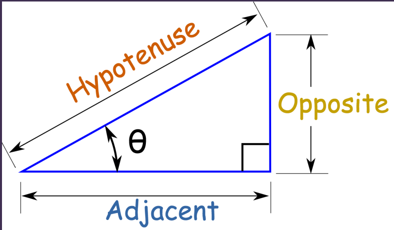


Trigonometry



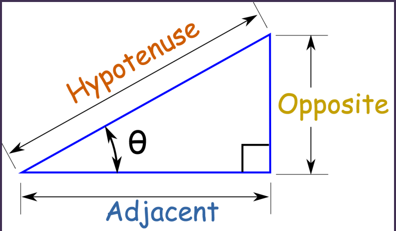
- $\sin\theta = \dfrac{\text{opposite}}{\text{hypotenuse}}$

Trigonometry



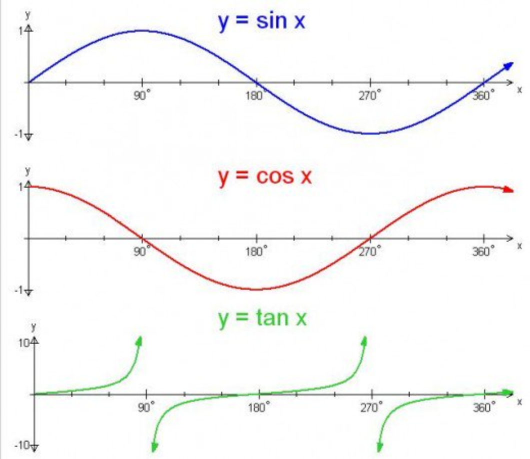
- $\sin\theta = \dfrac{\text{opposite}}{\text{hypotenuse}}$
- $\cos\theta = \dfrac{\text{adjacent}}{\text{hypotenuse}}$

Trigonometry



- $\sin\theta = \frac{\text{opposite}}{\text{hypotenuse}}$
- $\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}}$
- $\tan\theta = \frac{\text{opposite}}{\text{adjacent}}$

Sine, cosine and tangent



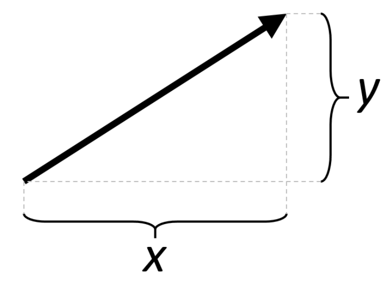
Magnitude and direction

Magnitude and direction

A vector has **components**

Magnitude and direction

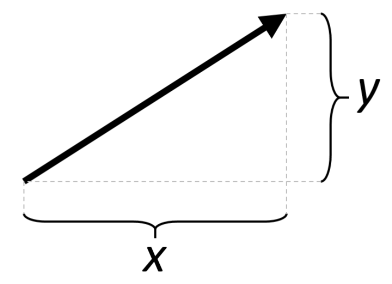
A vector has **components**



Magnitude and direction

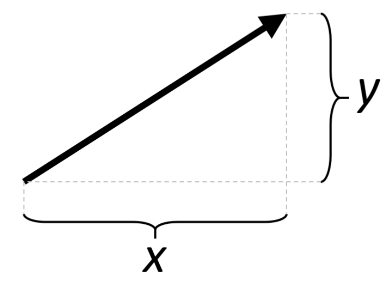
A vector also has **direction** and **magnitude** (or **length**)

A vector has **components**

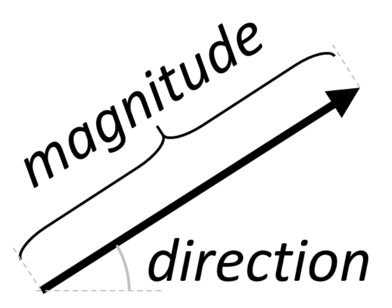


Magnitude and direction

A vector has **components**

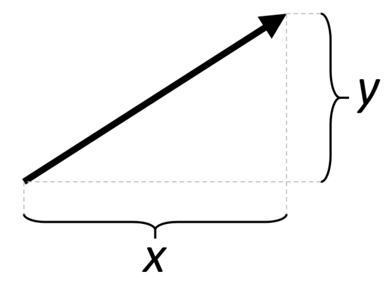


A vector also has **direction** and **magnitude** (or **length**)

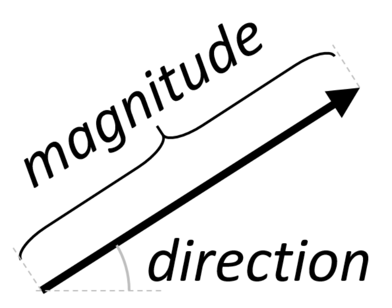


Magnitude and direction

A vector has **components**



A vector also has **direction** and **magnitude** (or **length**)



(Direction is measured as an angle from the positive *x*-axis)

Magnitude and direction

Magnitude and direction

- The magnitude of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\sqrt{x^2 + y^2}$

Magnitude and direction

- The magnitude of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\sqrt{x^2 + y^2}$
- The direction of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\tan^{-1}\left(\frac{y}{x}\right)$

Magnitude and direction

- The magnitude of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\sqrt{x^2 + y^2}$
- The direction of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\tan^{-1}\left(\frac{y}{x}\right)$
- The vector with magnitude r and direction θ is $\begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix}$

Magnitude and direction

- The magnitude of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\sqrt{x^2 + y^2}$
- The direction of $\begin{pmatrix} x \\ y \end{pmatrix}$ is $\tan^{-1}\left(\frac{y}{x}\right)$
- The vector with magnitude r and direction θ is $\begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix}$
- Multiplication: if u is a vector with magnitude r and direction θ, then $c \times u$ has magnitude $c \times r$ and direction θ

Worksheet D