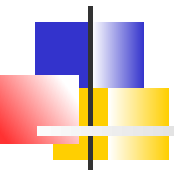
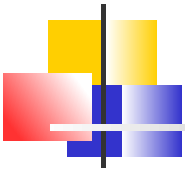
Software and Programming I

Arrays and Basic Algorithms



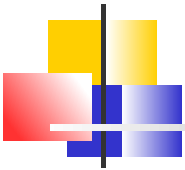
Roman Kontchakov / Carsten Fuhs

Birkbeck, University of London



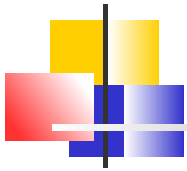
Outline

- Arrays
- Common Array Algorithms
- Enhanced `for` Loop
- Using Arrays with Methods
  - Sections 6.1–6.4
- slides are available at
  `www.dcs.bbk.ac.uk/~roman/sp1`



Operation Precedence

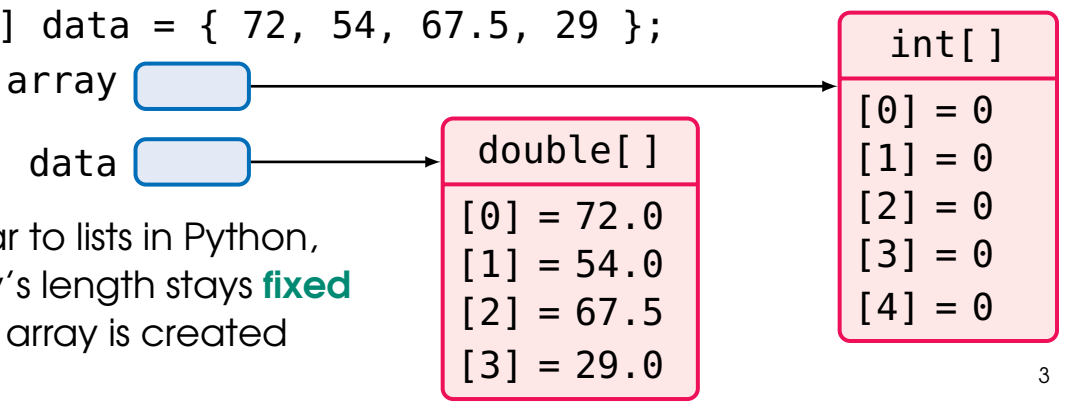
- `()` method call — highest
- `!`, `(type)` type cast, `++`, `--` — unary
- `*`, `/`, `%` — multiplicative
- `+`, `-` — additive
- `<`, `<=`, `>=`, `>` — relational
- `==`, `!=` — equality
- `&&` — logical AND
- `||` — logical OR
- `?:` — conditional
- `=`, `+=`, `...` — assignments — lowest



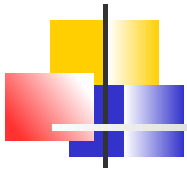
Arrays

An **array** collects a **sequence** of values of the same **type**

```
1 // empty array of 5 integers
2 //      0 is the default value
3 int[] array = new int[5];
4 // list of initial values
5 double[] data = { 72, 54, 67.5, 29 };
```



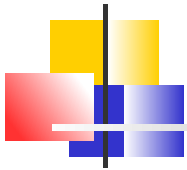
NB: similar to lists in Python, but array's length stays **fixed** after the array is created



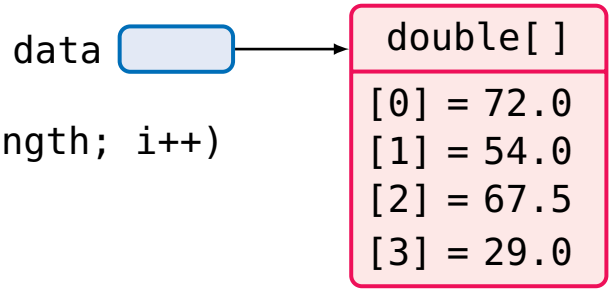
Array Elements

If `data` is a variable of an array type `someType[]`, then

- use the expression `data.length`
  to find the **number** of elements in an array `data`
  similar to `len(data)` for Python lists
- individual elements in an array are accessed by
  an integer index, using notation `data[i]`,
  where `i` is an integer expression
- the elements of arrays are numbered from **0**
  to `data.length - 1`
- an array element can be used in expressions
  like any other variable



Arrays: Example 1

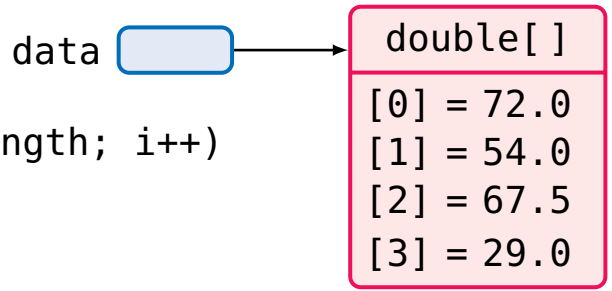



```
1 double min = data[0];
2 for (int i = 0; i < data.length; i++)
3     if (data[i] < min)
4         min = data[i];
```

min	i	data[i]	data[i] < min	min
72.0	0	72.0	false	72.0
72.0	1	54.0	true	54.0
54.0	2	67.5	false	54.0
54.0	3	29.0	true	29.0

NB: does not work correctly on empty `data` (if `data.length` is `0`)

Arrays: Example 2



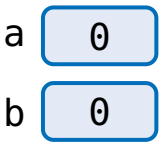

```
1 int mi = 0;
2 for (int i = 0; i < data.length; i++)
3     if (data[i] < data[mi])
4         mi = i;
```

mi	i	data[i]	data[mi]	data[i] < data[mi]	mi
0	0	72.0	72.0	false	0
0	1	54.0	72.0	true	1
1	2	67.5	54.0	false	1
1	3	29.0	54.0	true	3

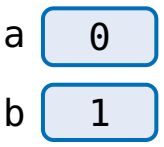
Primitive Datatypes: Values are Copied

```
1 int a = 0;
```


```
2 int b = a;
```




```
3 b++;
```



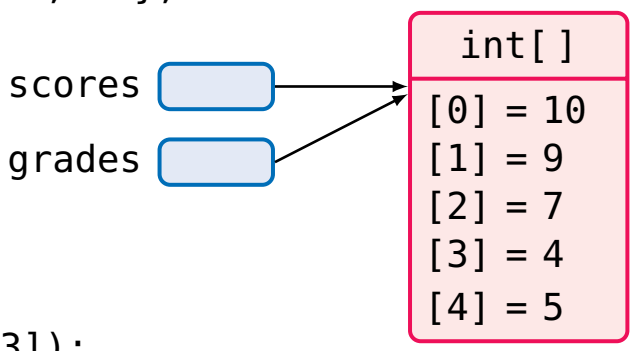
by executing `b++;` only `b` is changed, `a` remains the same

Arrays: References are Copied

an **array variable** specifies the location of an array

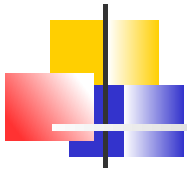
copying the reference yields another reference to the **same** array

```
1 int[] scores = { 10, 9, 7, 4, 5 };
2 int[] grades = scores;
```




```
3 scores[3] = 2;
4 // prints 2, not 4 !
5 System.out.println(grades[3]);
```

NB: just like lists in Python!



Arrays: Common Errors

- array `data` has elements
  with index values `0` to `data.length - 1` only

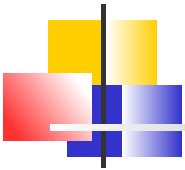
no negative indexes!

run-time error: `java.lang.ArrayIndexOutOfBoundsException`

- arrays must be initialised
  (otherwise the reference is `null`)

run-time error: `java.lang.NullPointerException`

NB: these run-time errors are Java **exceptions** (week 11)
if you do nothing to catch them, your program is simply terminated
like with `java.lang.ArithmeticException`: division by zero



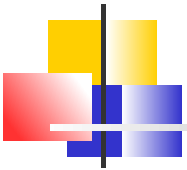
Algorithms: Average

```
1 double total = 0;
2 for (int i = 0; i < data.length; i++)
3     total += data[i];
4 double average = total / data.length;
```

what if `data.length == 0`?

floating-point division does not throw an exception, instead the result is NaN (not-a-number)

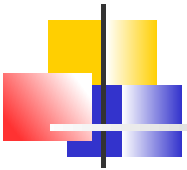
```
4 double average = 0;
5 if (data.length > 0)
6     average = total / data.length;
```



Algorithms: Element Separator

```
1 for (int i = 0; i < data.length; i++) {
2     // print comma except for the first element
3     if (i > 0)
4         System.out.print(", ");
5     System.out.print(data[i]);
6 }
```

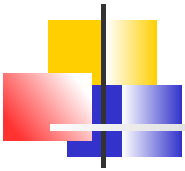
Exercise: modify the code so that the comma is printed *after* the current element (i.e., not before as in the code above)



Algorithms: Linear Search

```
1  int searchedValue = 100;
2  int pos = 0;
3  boolean found = false;
4  while (pos < data.length && !found) {
5      if (data[pos] == searchedValue)
6          found = true;
7      else
8          pos++;
9  }
10 if (found)
11     System.out.println("Found at position: " + pos);
12 else
13     System.out.println("Not found");
```

NB: not optimal if the array is **sorted**



Arrays and Methods (1)

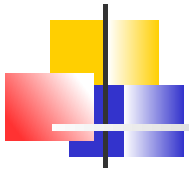
arrays can occur as method parameters. . .

```
1 public static double sum(double[] data) {
2     double total = 0;
3     for (int i = 0; i < data.length; i++) {
4         total += data[i];
5     }
6     return total;
7 }
```

NB: brackets can also be placed after the array's name

```
double data[]
```

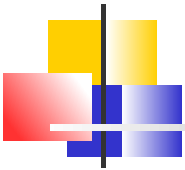
this, however, is not recommended —
the brackets identify the array type and should appear with the type designation



Enhanced For Loop

use the enhanced `for` loop to visit all elements of an array

```
1  public static double sum(double[] data) {
2      double total = 0;
3      for (double e: data) {
4      // for (int i = 0; i < data.length; i++) {
5          // double e = data[i];
6          total += e;
7          // total += e;
8      }
9      // }
10     return total;
11 }
```



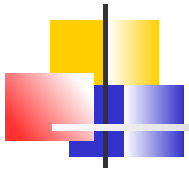
Arrays and Methods (2)

. . . and return values

```
1 public static int[] squares(int n) {
2     // operation new creates an array
3     int[] result = new int[n];
4     for (int i = 0; i < result.length; i++)
5         result[i] = i * i;
6     return result;
7 }
```

NB. the enhanced `for` loop cannot be used here

`e = i * i;` will **not** modify the array element



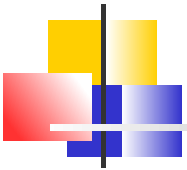
Arrays and Methods (3)

arrays are passed "**by reference**"

```
1 public static void multiply(double[] data,
2                              double factor) {
3     for (int i = 0; i < data.length; i++)
4         // changes the contents of the array
5         data[i] *= factor;
6 }
```

SO:

```
1 double[] myData = { 1, 2, 3 };
2 multiply(myData, 20);
3 System.out.println(myData[2]); // prints 60.0, not 3.0
```



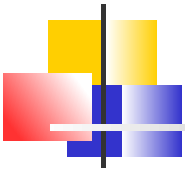
Fibonacci Numbers: Maths

$f_0 = 0$

$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$, for $n > 1$

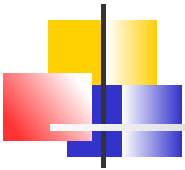
n	0	1	2	3	4	5	6	7	8
f_n	0	1	1	2	3	5	8	13	21



Fibonacci Numbers with Arrays

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \text{ for } n > 1$$

```
1    public static int[] fibonacci(int k) {
2        int[] f = new int[k];
3        f[0] = 0;
4        f[1] = 1;
5        for (int n = 2; n < k; n++) {
6            f[n] = f[n-1] + f[n-2];
7        }
8        return f;
9    }
```



Take Home Messages

- an array is a sequence of values of the same type
- an array index in an array `data` must be
  `>= 0` and `< data.length`
- an array variable specifies the location of an array; copying the reference yields a second reference to the same array
- arrays can occur as method parameters and return values (passing "by reference")