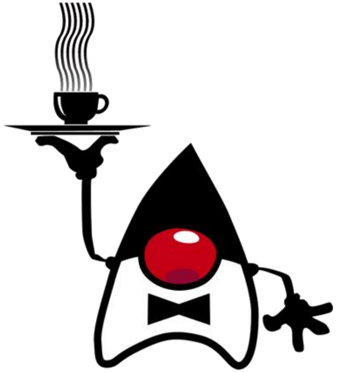
Data Types and Operators



CO7005 Software Development Techniques

Dr Stuart Cunningham
s.cunningham@chester.ac.uk

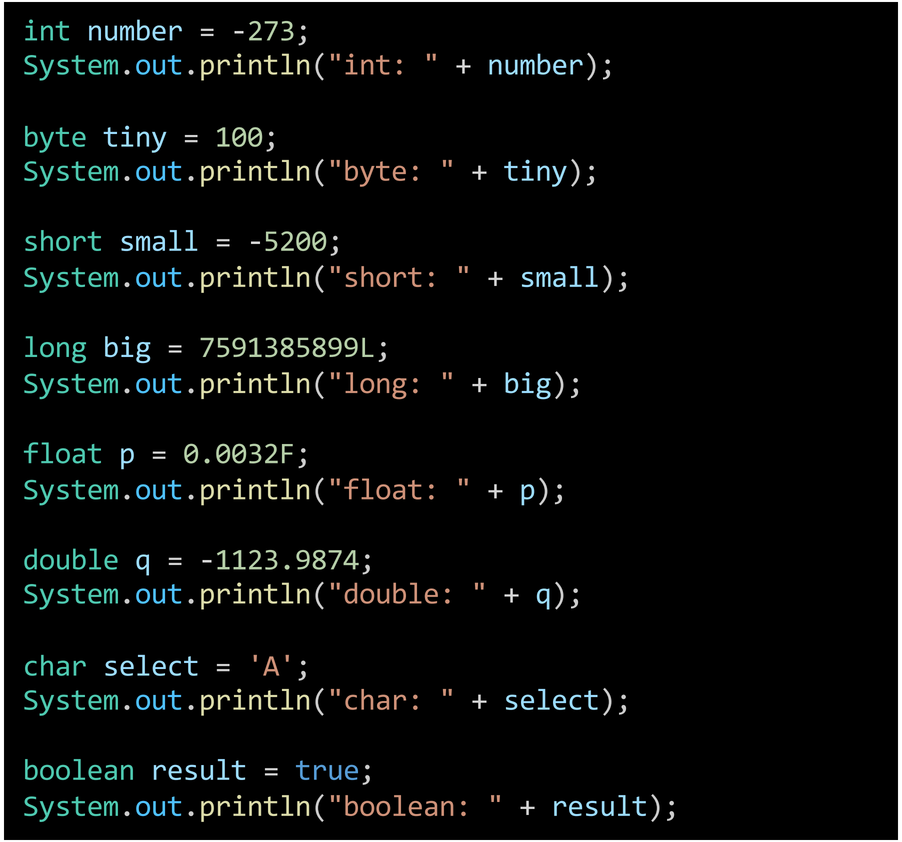
Recap

- Core Input-Process-Output model
- Fundamentals of arithmetic
- Use of `int` and `String` variables
- Simple input and output

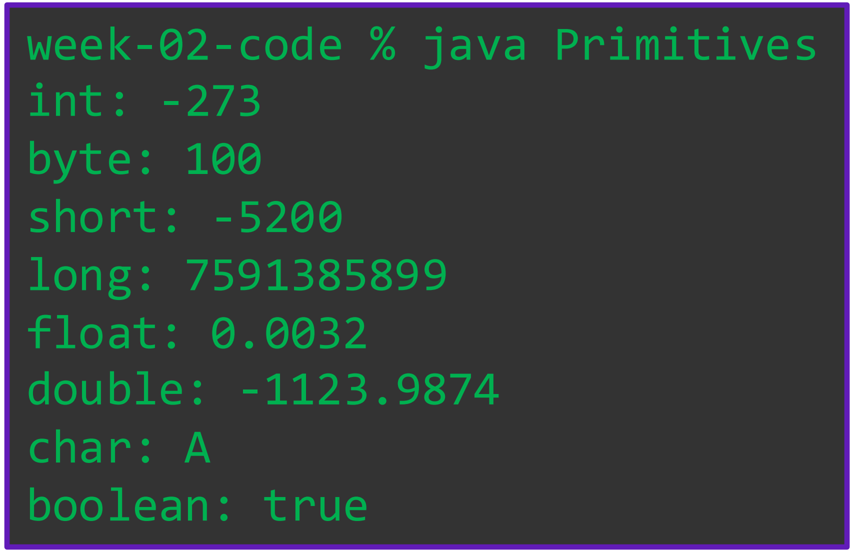
Java’s Eight Primitive Data Types

Type	Description	Example
`int`	Integer between -2147483648 and 2147483647	`int number = -273;`
`byte`	Integer between -128 and 127	`byte tiny = 100;`
`short`	Integer between -32768 and 32767	`short small = -5200;`
`long`	Integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	`long big = 7591385899L;`
`float`	Floating point number with *big* range	`float p = 0.0032F;`
`double`	Floating point number with *huge* range	`double q = -1123.9874;`
`char`	Single Unicode alphanumeric character	`char select = 'A';`
`boolean`	Boolean. False (0) or True (1)	`boolean result = true;`

Java’s Eight Primitive Data Types



```java
int number = -273;
System.out.println("int: " + number);

byte tiny = 100;
System.out.println("byte: " + tiny);

short small = -5200;
System.out.println("short: " + small);

long big = 7591385899L;
System.out.println("long: " + big);

float p = 0.0032F;
System.out.println("float: " + p);

double q = -1123.9874;
System.out.println("double: " + q);

char select = 'A';
System.out.println("char: " + select);

boolean result = true;
System.out.println("boolean: " + result);
```



```
week-02-code % java Primitives
int: -273
byte: 100
short: -5200
long: 7591385899
float: 0.0032
double: -1123.9874
char: A
boolean: true
```

The String Variable

- The `char` variable only stores a single character
- To store a sequence of characters, use `String`
- Not a primitive data type, but an *object*
  - Indicated by use of a capital ‘S’ at declaration
- Otherwise, it works a lot like the other variables so far

```
String myString = "The quick brown fox jumps over the lazy dog.";
```

Variables and the Scanner (input) Class

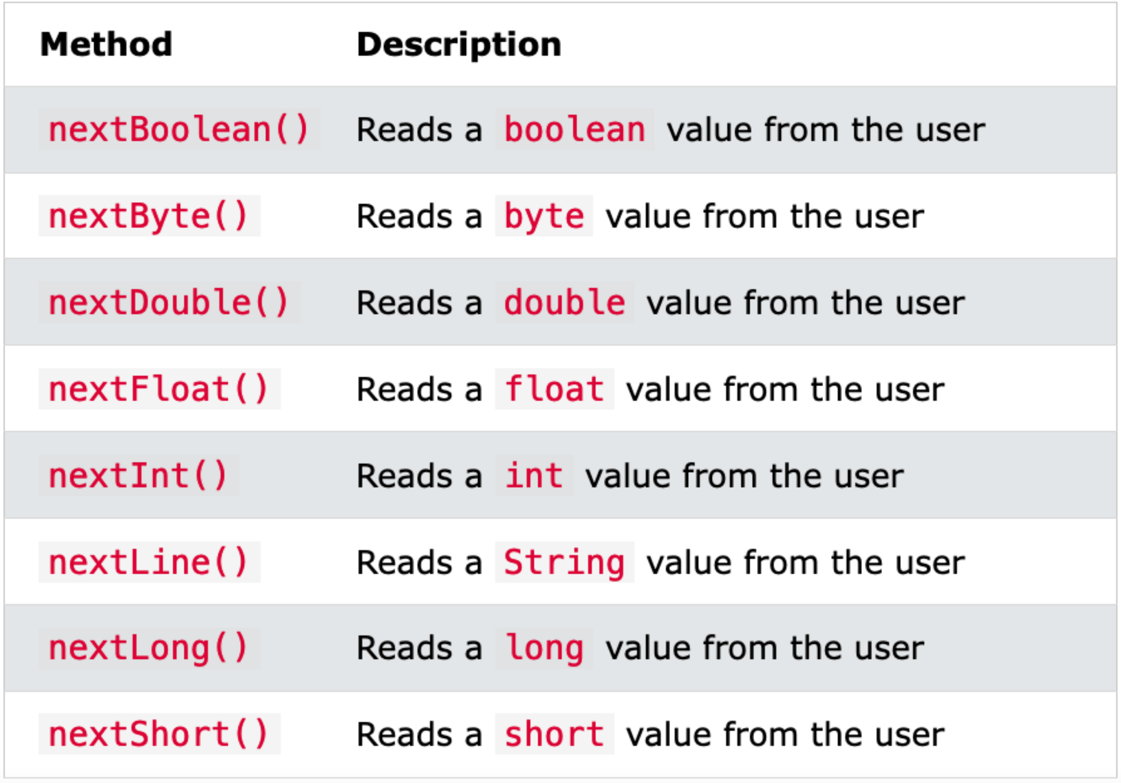
- Use the Scanner to get input from the user/terminal
  - Different methods depending on variable type
- In most cases this is `.next<VariableType>()`
- For example:

```
int number = input.nextInt();
float p = input.nextFloat();
boolean result = input.nextBoolean();
```

- Strings and chars are a little different

```
String myString = input.nextLine();
char select = input.next().charAt(0);
```

Variables and the Scanner (input) Class



Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

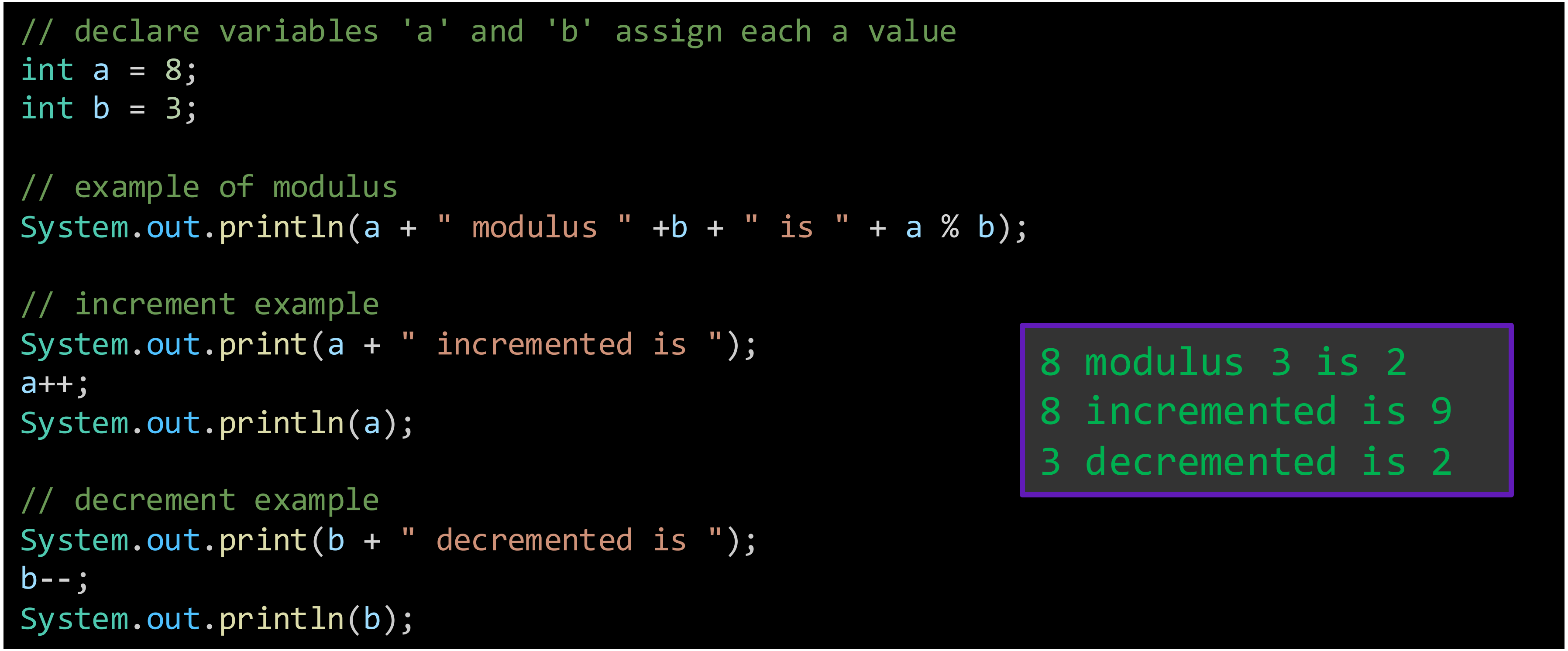
Source: https://www.w3schools.com/java/java_user_input.asp

More Operators

- As well as the four arithmetic operators (+, -, *, /) there are others that are useful when writing programs

Operator	Operation	Example
%	Modulus (remainder)	a % b
++	Increment (add 1)	a++ or ++a
--	Decrement (subtract 1)	b-- or --b
+= n	Increment by n	a += 3
-= n	Decrement by n	b -+5

More Operators



```java
// declare variables 'a' and 'b' assign each a value
int a = 8;
int b = 3;

// example of modulus
System.out.println(a + " modulus " +b + " is " + a % b);

// increment example
System.out.print(a + " incremented is ");
a++;
System.out.println(a);

// decrement example
System.out.print(b + " decremented is ");
b--;
System.out.println(b);
```

```
8 modulus 3 is 2
8 incremented is 9
3 decremented is 2
```

Relational Operators

- Make comparisons between numeric variable values
  - Useful to make decisions or control program flow
- Are logical, always return a `boolean` outcome

Operator	Description	Example
`==`	Equal to	`a==b`
`!=`	Not equal to	`a!=b`
`>`	Greater than	`a>b`
`>=`	Greater than or equal to	`a>=b`
`<`	Less than	`a<b`
`<=`	Less than or equal to	`a<=b`

Relational Operators

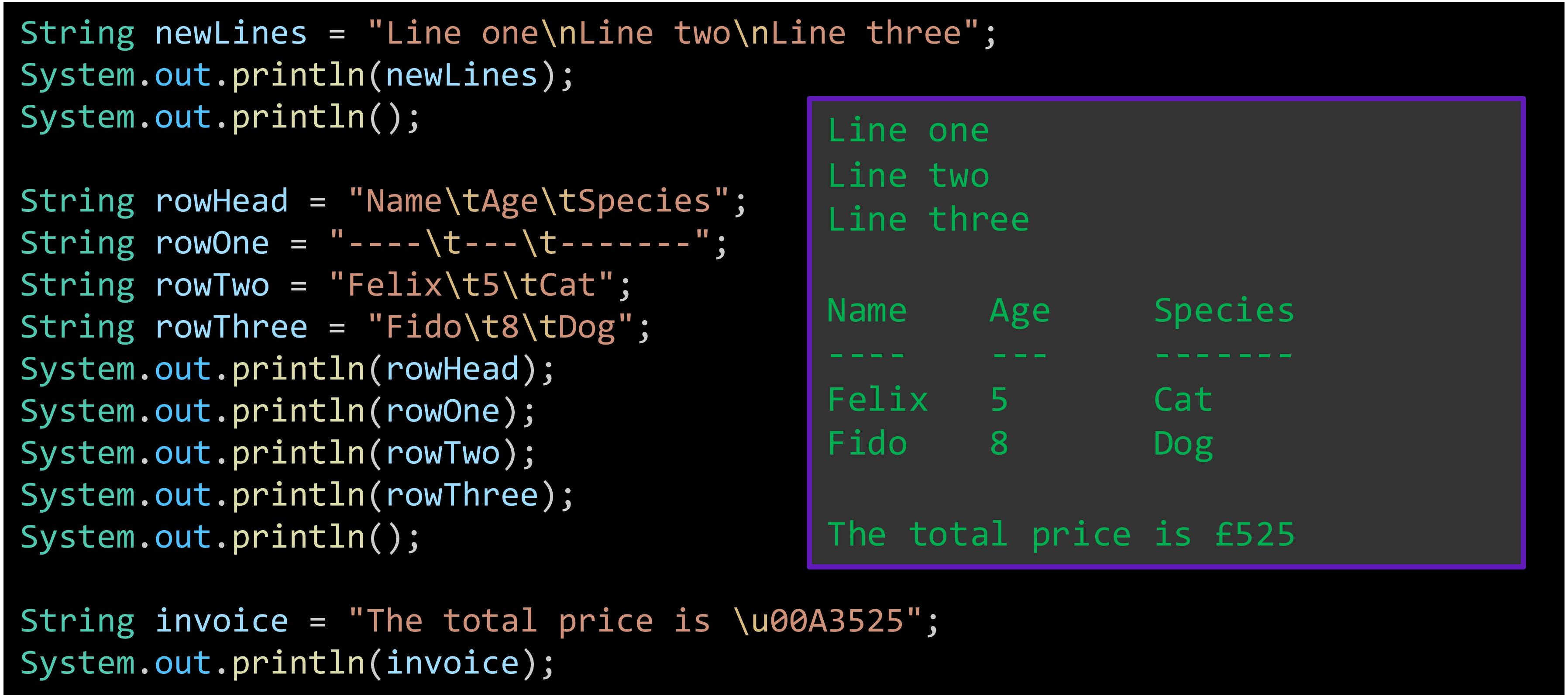
```
int x = 15, y = 20, z = 15;

System.out.println("x equal to y? "+(x==y));
System.out.println("x NOT equal to y? "+(x!=y));
System.out.println("x greater than y? "+(x>y));
System.out.println("z greater than or equal to y? "+(z>=y));
System.out.println("x less than y? "+(x<y));
System.out.println("x less than or equal to z? "+(x<=z));
```

```
x equal to y? false
x NOT equal to y? true
x greater than y? false
z greater than or equal to y? false
x less than y? true
x less than or equal to z? true
```

Strings and Escape Sequences

- *Escape sequences*, perform operations or add special characters to text
- Preceded by the backslash (\) character
- Available in `String` or `char` variables
- Or directly in `System.out.print("")`

StringEscape

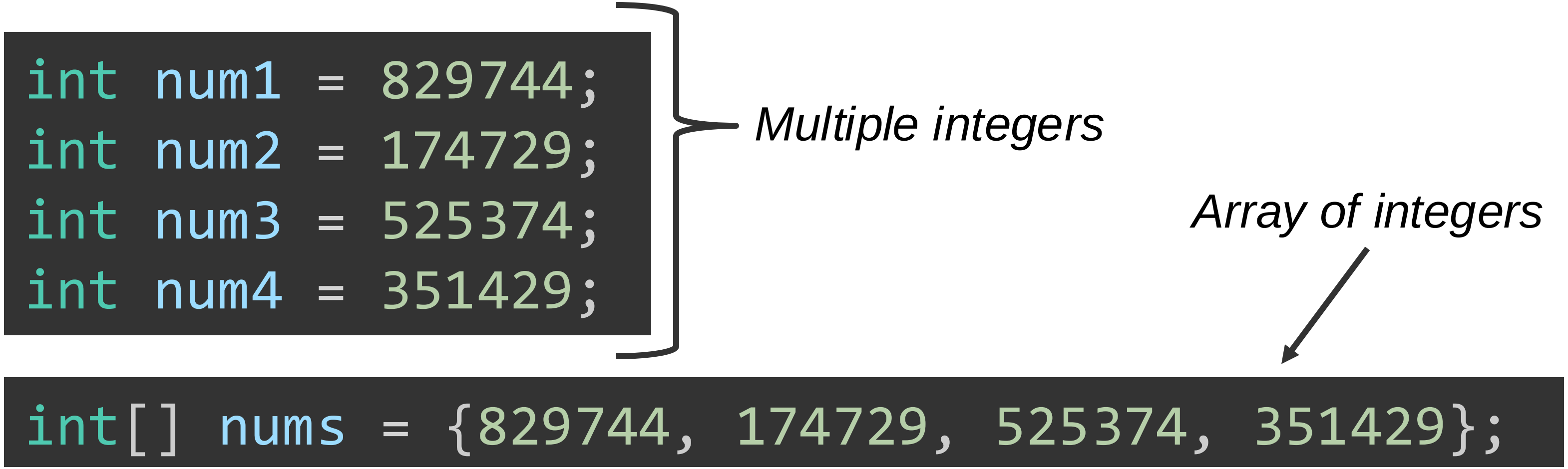


```
String newLines = "Line one\nLine two\nLine three";
System.out.println(newLines);
System.out.println();

String rowHead = "Name\tAge\tSpecies";
String rowOne = "----\t---\t-------";
String rowTwo = "Felix\t5\tCat";
String rowThree = "Fido\t8\tDog";
System.out.println(rowHead);
System.out.println(rowOne);
System.out.println(rowTwo);
System.out.println(rowThree);
System.out.println();

String invoice = "The total price is \u00A3525";
System.out.println(invoice);
```

```
Line one
Line two
Line three

Name    Age     Species
----    ---     -------
Felix   5       Cat
Fido    8       Dog

The total price is £525
```

Arrays

- Arrays are a special type of variable (and *objects*)
- Enable storage of a *sequence* of values of the same type
- Consider a list of telephone numbers…



```
int num1 = 829744;
int num2 = 174729;
int num3 = 525374;
int num4 = 351429;
```

Multiple integers

Array of integers

```
int[] nums = {829744, 174729, 525374, 351429};
```

Arrays

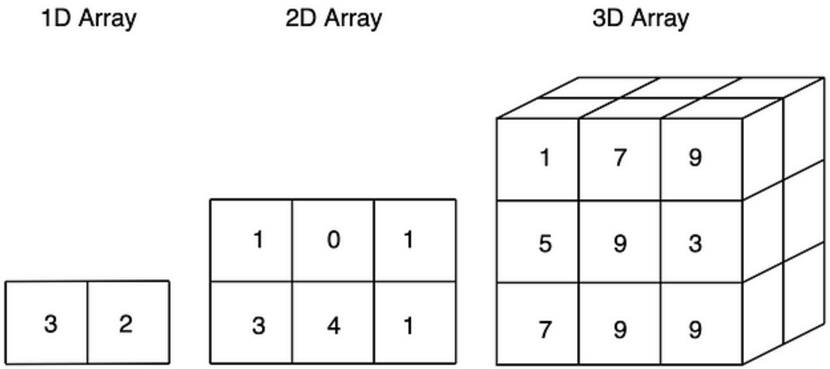
- Each item in an array is an *element*
- Elements are accessed using an *index* (in square brackets [ ])
- Indices begin at zero, the n^{th} element has index *n-1*
- E.g., display the second item in the telephone numbers list…

```
int[] nums = {829744, 174729, 525374, 351429};
System.out.println("Number 2 is: " + nums[1]);
```

```
Number 2 is: 174729
```

Multi-Dimensional Arrays

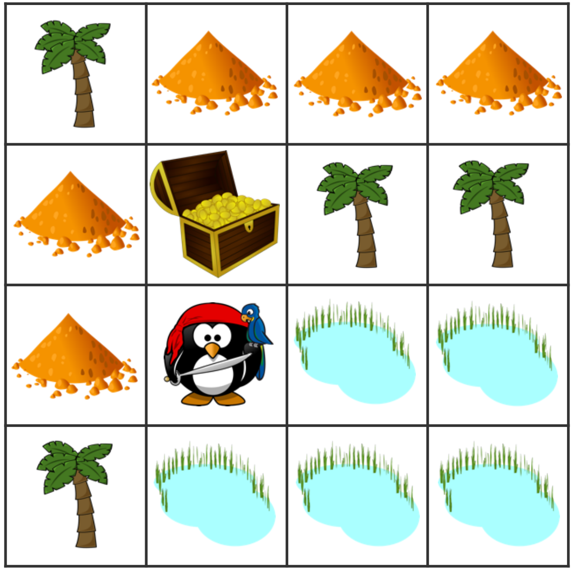
- Previous examples are one-dimensional arrays (e.g., a list)
- Arrays with higher dimensions are common, but especially those with two-dimensions
- Two-dimensional arrays represent tables, maps, etc.
- Three-dimensional arrays represent multiple tables, a 3D map or object, etc.



Source: https://towardsdatascience.com/numpy-array-cookbook-generating-and-manipulating-arrays-in-python-2195c3988b09

Multi-Dimensional Arrays: 2D

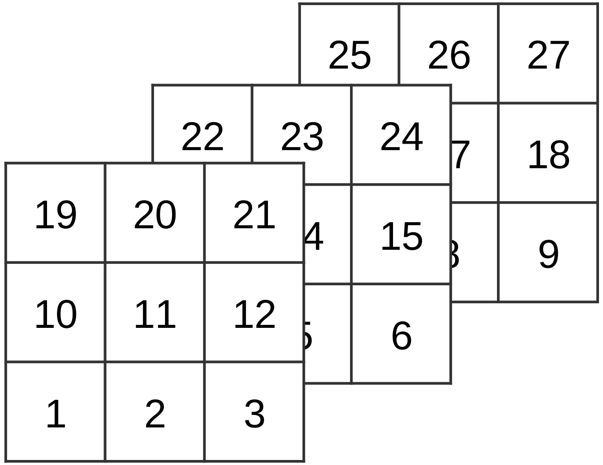
- Implemented by embedding an array-within-an-array
- Consider representing a map, with a 2D array of characters

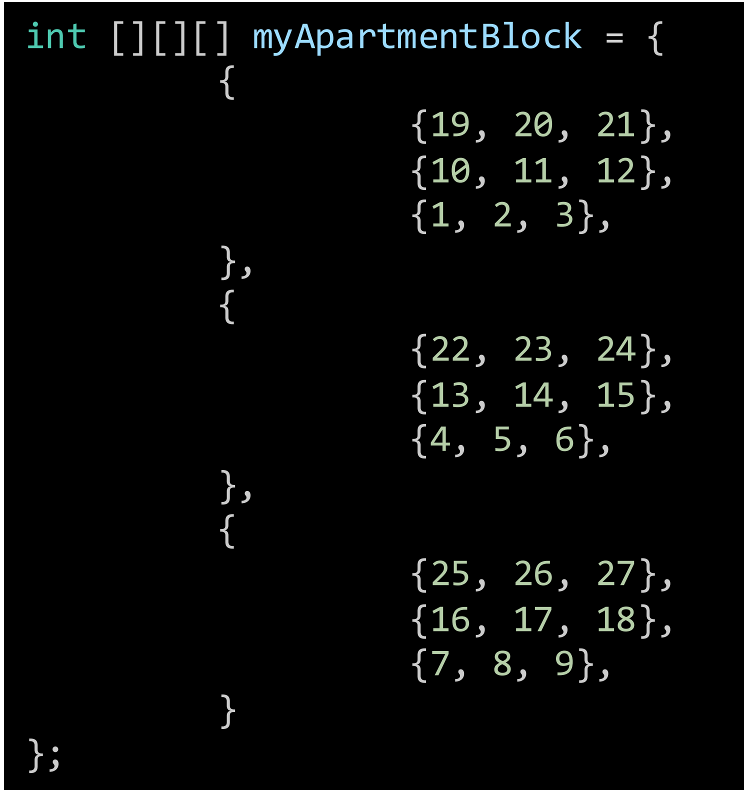


```
// key for map elements...
// S = Sand; X = Treasure;
// W = Water; P = Pirate; T = Tree

char [][] myTreasureMap = {
       {'T', 'S', 'S', 'S’},
       {'S', 'X', 'T', 'T’},
       {'S', 'P', 'W', 'W’},
       {'T', 'W', 'W', 'W'},
};
```

Multi-Dimensional Arrays: 3D

- Extends the process of the 2D array
- Consider a block of apartments, using a 3D array of integers





```
int [][][] myApartmentBlock = {
         {
                   {19, 20, 21},
                   {10, 11, 12},
                   {1, 2, 3},
         },
         {
                   {22, 23, 24},
                   {13, 14, 15},
                   {4, 5, 6},
         },
         {
                   {25, 26, 27},
                   {16, 17, 18},
                   {7, 8, 9},
         }
};
```

Efficiency and Data Types

- Simple rule: don’t use a variable type that you don’t need
  - But some Java functions require a specific type – watch for errors
- Larger types use more memory
  - A `double` uses 64 bits, while `float` and `int` use 32 bits
  - E.g., `int` calculations are faster than `float` or `double`
- Remember to close the Scanner when done with it
  - `input.close(); //assuming instance named ‘input’`
- Each dimension of an array (potentially) adds an order of magnitude of complexity to work with it