A.STEFANCHUK IT STEP

DIGGING FURTHER

DART INTRO, LAYOUT WIDGETS COMBINED, WRITING LOGIC

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

- Dart can understand what types of variable it stores (var)

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

- Dart can understand what types of variable it stores (var)
- Object is any type - so you will need to cast it later

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

- Dart can understand what types of variable it stores (var)
- Object is any type - so you will need to cast it later
- You can specify variable type yourself

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

```
void sayHello(String name) {
  print('Hello, $name');
}
```

VARIABLES

```
var name = 'Andriy';

Object name2 = 'Andy';

String name3 = 'Andrew';
```

```
sayHello(name);
sayHello(name2 as String);
sayHello(name3);
```

```
void sayHello(String name) {
  print('Hello, $name');
}
```

VARIABLES

Everything you can put inside a variable is an object

VARIABLES

Everything you can put inside a variable is an object

That means that all numbers, bools, functions are objects

VARIABLES

Everything you can put inside a variable is an object

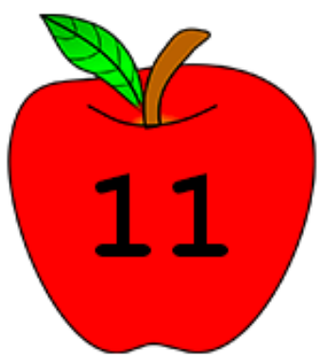
That means that all numbers, bools, functions are objects

Object is base object for all classes

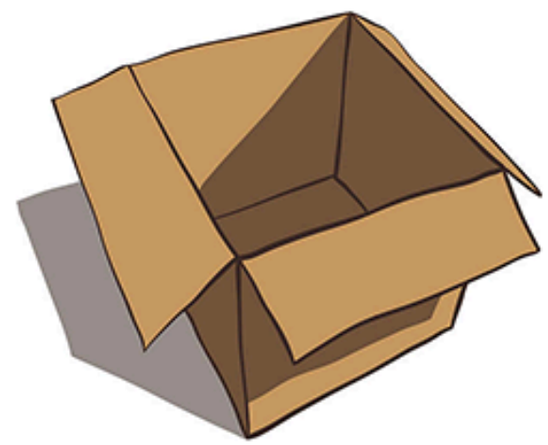
BASE DART TYPES

```
int simpleNumber = 12;
double decimalNumber = 13.4;
String text = 'Test string';
bool isEnabled = true;

List names = ['Andriy', 'Masha', 'Nastya'];
Set movies = {'Infinity war', 'Endgame'};
Map numbers = {'one': 1, 'two': 2};

int? optionalNumber = null;
```

OPTIONAL TYPE



Int



Int?

OTHER VARIABLES KEYWORDS

```
late String description;

final name = 'Bob';
name = 'Tommy';

var name2 = 'Billy';
name2 = 'James';

const PI = 3.14;
const double something;
```

OTHER VARIABLES KEYWORDS

```
late String description;

final name = 'Bob';
name = 'Tommy';

var name2 = 'Billy';
name2 = 'James';

const PI = 3.14;
const double something;
```

- Can be given value later

OTHER VARIABLES KEYWORDS

```
late String description;

final name = 'Bob';
name = 'Tommy';

var name2 = 'Billy';
name2 = 'James';

const PI = 3.14;
const double something;
```

- Can be given value later
- Can be given value only one time

OTHER VARIABLES KEYWORDS

```
late String description;

final name = 'Bob';
name = 'Tommy';

var name2 = 'Billy';
name2 = 'James';

const PI = 3.14;
const double something;
```

- Can be given value later
- Can be given value only one time
- Must have value at compile time (colors, constants)

FUNCTIONS

```
void simpleFunction() {
  print('I do just this');
}
```

FUNCTIONS

Return type

```
void simpleFunction() {
  print('I do just this');
}
```

Body

FUNCTIONS

```
int sum(int a, int b) {
  return a + b;
}

var result = sum(10, 13);
print(result);
```

FUNCTIONS

Parameters list

```
int sum(int a, int b) {
  return a + b;
}

var result = sum(10, 13);
print(result);
```

Function call

FUNCTIONS

- Named parameters

```
double calculateBodyMassIndex({required double height, required double weight}) {
  return height * weight * 1.2;
}

var result2 = calculateBodyMassIndex(height: 177.7, weight: 68.2);
print(result2);
```

FUNCTIONS

- Named parameters

```
double calculateBodyMassIndex({required double height, required double weight}) {
  return height * weight * 1.2;
}

var result2 = calculateBodyMassIndex(height: 177.7, weight: 68.2);
print(result2);
```

```
double calculateBodyMassIndex(double height, {double weight = 60}) {
  return height * weight * 1.2;
}

var result2 = calculateBodyMassIndex(177.7, weight: 68.2);
print(result2);
```

FUNCTIONS

- Optional parameters

```
String say(String from, String msg, [String? device]) {
  var result = '$from says $msg';
  if (device != null) {
    result = '$result with a $device';
  }
  return result;
}

say('one', 'two');
```

ARITHMETIC OPERATORS

```
assert(2 + 3 == 5);
assert(2 - 3 == -1);
assert(2 * 3 == 6);
assert(5 / 2 == 2.5); // Result is a double
assert(5 ~/ 2 == 2); // Result is an int
assert(5 % 2 == 1); // Remainder
```

ARITHMETIC OPERATORS

```
assert(2 + 3 == 5);
assert(2 - 3 == -1);
assert(2 * 3 == 6);
assert(5 / 2 == 2.5); // Result is a double
assert(5 ~/ 2 == 2); // Result is an int
assert(5 % 2 == 1); // Remainder
```

Operator	Meaning
`++var`	`var = var + 1` (expression value is `var + 1`)
`var++`	`var = var + 1` (expression value is `var`)
`--var`	`var = var - 1` (expression value is `var - 1`)
`var--`	`var = var - 1` (expression value is `var`)

EQUALITY OPERATORS

==	Equal; see discussion below
! =	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

ASSIGNMENT OPERATORS

```
var a = 2; // Assign using =
a *= 3; // Assign and multiply: a = a * 3
assert(a == 6);
```

LOGICAL OPERATORS

Operator	Meaning
`!expr`	inverts the following expression (changes false to true, and vice versa)
`||`	logical OR
`&&`	logical AND

```
if (!done && (col == 0 || col == 3)) {
  // ...Do something...
}
```

FLOW CONTROL OPERATORS

- If-else

```
if (isRaining()) {
  you.bringRainCoat();
} else if (isSnowing()) {
  you.wearJacket();
} else {
  car.putTopDown();
}
```

FLOW CONTROL OPERATORS

- For loop

```
var message = StringBuffer('Dart is fun');
for (var i = 0; i < 5; i++) {
  message.write('!');
}
```

FLOW CONTROL OPERATORS

- For loop

```
var message = StringBuffer('Dart is fun');
for (var i = 0; i < 5; i++) {
  message.write('!');
}
```

```
for (final candidate in candidates) {
  candidate.interview();
}
```

FLOW CONTROL OPERATORS

- While and do while

```
while (!isDone()) {
  doSomething();
}
```

FLOW CONTROL OPERATORS

- While and do while

```
while (!isDone()) {
  doSomething();
}
```

```
do {
  printLine();
} while (!atEndOfPage());
```

FLOW CONTROL OPERATORS

- Break and continue

```
while (true) {
  if (shutDownRequested()) break;
  processIncomingRequests();
}
```

FLOW CONTROL OPERATORS

- Break and continue

```
while (true) {
  if (shutDownRequested()) break;
  processIncomingRequests();
}
```

```
for (int i = 0; i < candidates.length; i++) {
  var candidate = candidates[i];
  if (candidate.yearsExperience < 5) {
    continue;
  }
  candidate.interview();
}
```

FLOW CONTROL OPERATORS

- Switch-case

```
var command = 'OPEN';
switch (command) {
  case 'CLOSED':
    executeClosed();
    break;
  case 'PENDING':
    executePending();
    break;
  case 'APPROVED':
    executeApproved();
    break;
  case 'DENIED':
    executeDenied();
    break;
  case 'OPEN':
    executeOpen();
    break;
  default:
    executeUnknown();
}
```