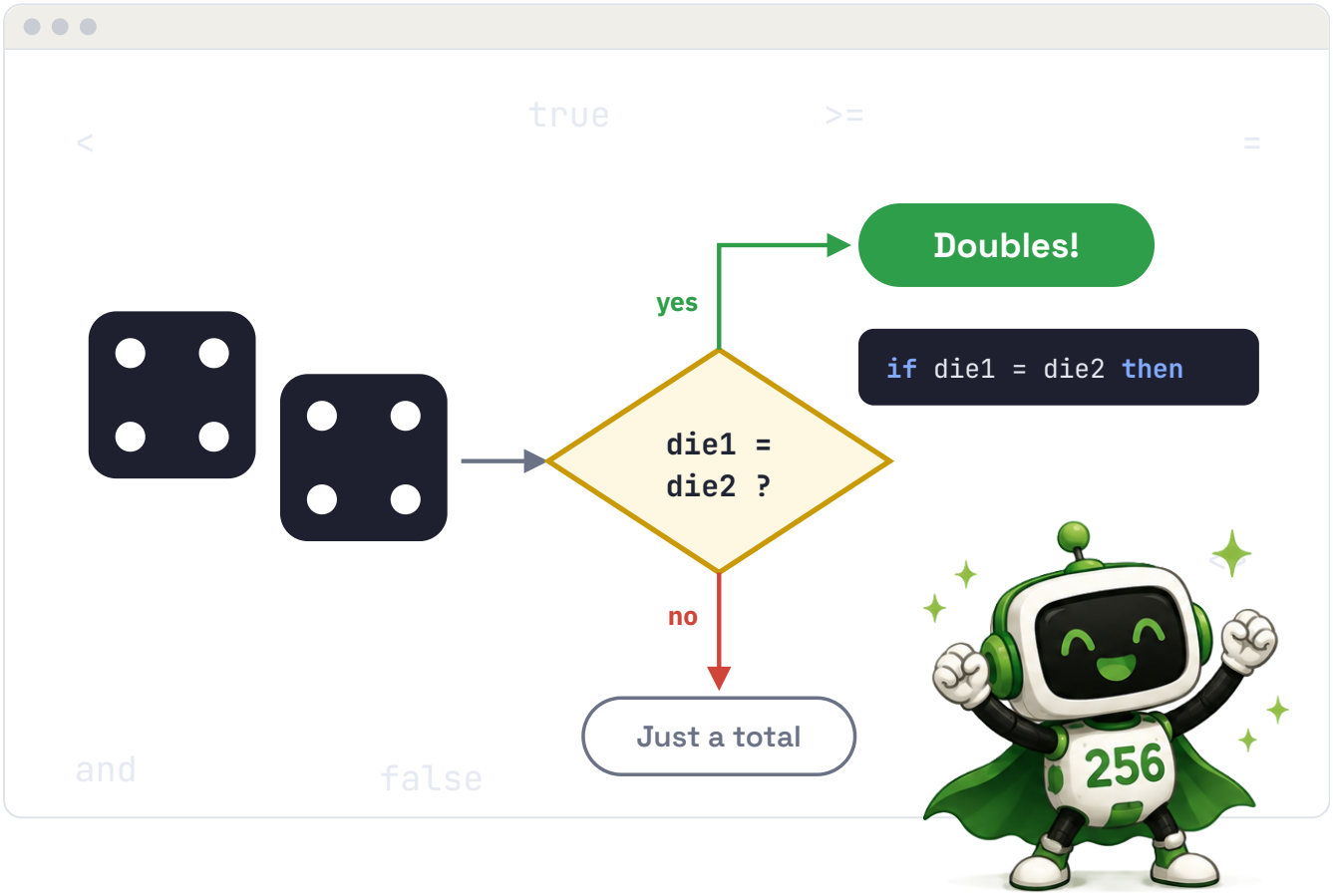
LEARN TO CODE · DECISIONS

Chapter 7

Decisions, Decisions, Decisions

Computers are whizzes at comparing things. In this chapter you learn how to compare two values, how to join comparisons together, and how to make your program do something only when a comparison is true. You will also roll dice and flip coins with random numbers.



© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 1

True, false and comparing

Before a program can decide anything, it has to ask a question that has a yes-or-no answer.

BASIC-256 has one more special kind of value: the **Boolean** value, named after the mathematician George Boole. A Boolean value is either *true* or *false*. You will rarely type one yourself — they are what you get when you compare two things.



NEW CONCEPT

```
true
false
```

The two Boolean constants **true** and **false** can be used in any expression. BASIC-256 stores **true** as the integer **1** and **false** as the integer **0**, so `print true` prints `1`.

Comparison operators

You already know the arithmetic operators `+`, `-`, `*` and `/`. A **comparison operator** also works with two values, but instead of calculating a new number it answers a question: is it true? The answer is always *true* (1) or *false* (0).

OPERATOR	IS IT TRUE THAT THE LEFT SIDE IS …	EXAMPLE	RESULT
`<`	less than	`3 < 5`	true
`<=`	less than or equal to	`5 <= 5`	true
`>`	greater than	`2 > 7`	false
`>=`	greater than or equal to	`7 >= 2`	true
`=`	equal to	`4 = 4`	true
`<>`	not equal to	`4 <> 4`	false

The six comparison operators. The results were checked in BASIC-256 2.3.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



NEW CONCEPT

```
expression1 operator expression2
```

The six comparison operators are: less than (`<`), less than or equal to (`<=`), greater than (`>`), greater than or equal to (`>=`), equal to (`=`) and not equal to (`<>`). They compare numbers and strings.

Strings are compared one character at a time, from the left. Between letters of the same kind that is alphabetical order, but **all capital letters come before lower case letters**, so `"Zebra" < "apple"` is true.



TRY IT

Print a few comparisons and predict the answer before you press Run: `print "cat" < "dog"` , `print "cat" < "Dog"` and `print "10" < "9"` . The second one is **0** — all capital letters come before lower case letters. The third one is **1**: as strings, “10” starts with a 1, and 1 comes before 9. Compare numbers as numbers (`10 < 9` is 0) when you mean numbers.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 2

Making simple decisions — the if statement

The `if` statement runs a statement only when a condition is true.

Now we can put the comparisons to work. The next program asks for your age and the age of a friend, and uses three **if** statements to tell you whether you are younger, the same age, or older.

compareages.kbs — BASIC-256

```
# compareages.kbs - compare two ages
inputinteger "How old are you? ", yourage
inputinteger "How old is your friend? ", friendage

print "You are ";
if yourage < friendage then print "younger than";
if yourage = friendage then print "the same age as";
if yourage > friendage then print "older than";
print " your friend."
```

TEXT OUTPUT

```
How old are you? 13
How old is your friend? 12
You are older than your friend.

How old are you? 12
How old is your friend? 12
You are the same age as your friend.
```

Program 37 · Compare Two Ages

Only one of the three conditions can be true, so only one of the three middle words is printed. The semicolons at the end of the **print** statements keep everything on one line, as you saw in Chapter 3.

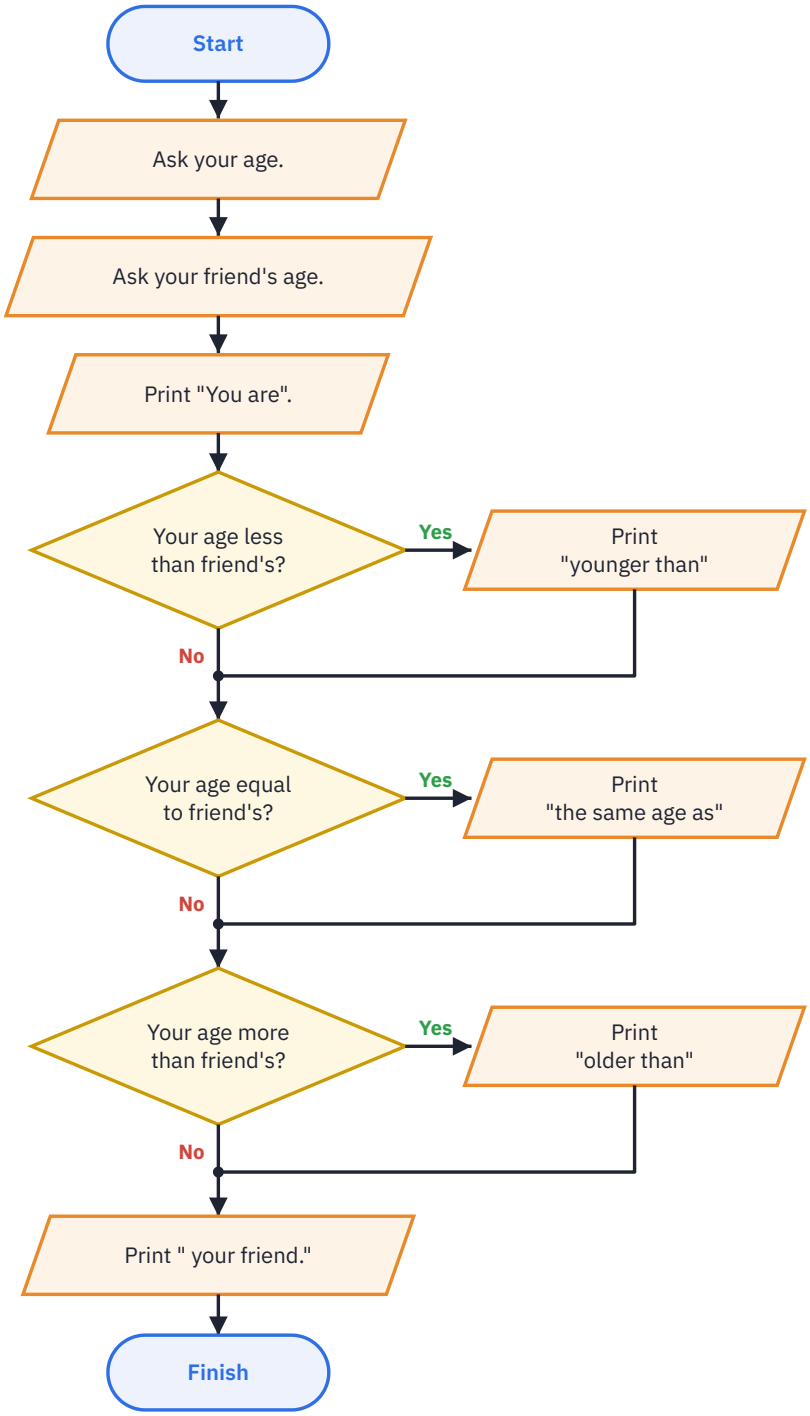


NEW CONCEPT

```
if condition then statement
```

If the condition is *true*, the statement after **then** is executed. If it is *false*, BASIC-256 skips it and goes on to the next line.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



Flowchart of Program 37. Each diamond is one **if** statement: the program always asks all three questions, but at most one answer is yes.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 3

Random numbers

Games and simulations need dice, coins and spinners. BASIC-256 has a random number generator built in.



NEW CONCEPT

```
rand
rand()
```

Every time **rand** is used in an expression it gives a new random number from 0 up to, but never including, 1 (0 ≤ n < 1). The brackets are optional.

To get a whole number from 1 to *r*, use `int(rand * r) + 1` — for a die, `int(rand * 6) + 1`.

A coin has two sides, so we can say that a number below 0.5 is heads and anything else is tails.

coinflip.kbs — BASIC-256

```
# coinflip.kbs - flip a coin
coin = rand
if coin < 0.5 then print "Heads."
if coin >= 0.5 then print "Tails."
```

TEXT OUTPUT

```
Heads.
Tails.
```

Program 38 · Coin Flip

Two runs. Yours will be different — that's the point.

Watch out — a logic error. You may be tempted to leave out the variable and write `if rand < 0.5 then print "Heads."` and `if rand >= 0.5 then print "Tails."`. That program has no syntax error, but sometimes it prints both words and sometimes neither, because each **rand** gives a *different* number. A mistake like this, where the program runs but does the wrong thing, is called a **logic error**. Pick the random number once, store it, and then test the variable.

The same "random" numbers every time

The numbers from **rand** are not really random: they are worked out by a formula from a starting value called the **seed**. When a program starts, BASIC-256 picks a seed from the clock, so every run is different. When you are

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

testing a program, or want a game level to come out the same every time, you can choose the seed yourself with the **seed** statement.

samerolls.kbs BASIC-256

```
# samerolls.kbs - the same "random" rolls every run
seed 2026
print "Roll 1: " + (int(rand * 6) + 1)
print "Roll 2: " + (int(rand * 6) + 1)
print "Roll 3: " + (int(rand * 6) + 1)
print "Roll 4: " + (int(rand * 6) + 1)
```

TEXT OUTPUT

```
Roll 1: 2
Roll 2: 1
Roll 3: 1
Roll 4: 5
```

```
Roll 1: 2
Roll 2: 1
Roll 3: 1
Roll 4: 5
```

Program 39 · The Same Rolls Every Time

Run twice, with the same result. Change 2026 to another number and you get different rolls — but again the same ones on every run.



NEW CONCEPT

```
seed number
```

The **seed** statement restarts the random number generator from the number you give. After the same seed, **rand** gives the same sequence of numbers every time the program is run, which makes a program with random numbers easy to test.

The sequence for a seed can be different on another kind of computer or in the browser version of BASIC-256, but it stays the same on every run on that computer.

Coming later: two more kinds of random. BASIC-256 has two other random number generators. **Gaussian** random numbers (the `ma_gaussian` function in the `math` module) cluster around an average, like people's heights. **OpenSimplex noise** (the `noise` function) gives smooth random values that change gradually — just right for landscapes, clouds and wandering paths. A seed fixes the noise pattern too. Both are covered in the **Advanced** chapters.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 4

Logical operators

Sometimes one comparison isn't enough. Logical operators join comparisons together.

BASIC-256 has four logical operators: **and**, **or**, **xor** and **not**. They work much like the same words in English — except that **or** means "one or the other *or both*". The Big Program at the end of this chapter uses `roll >= 4 and roll <= 6` to ask whether a die shows 4, 5 or 6.

OPERATOR	RESULT
`and`	*true* only if both sides are true.
`or`	*true* if either side is true, or both.
`xor`	*true* if exactly one side is true — "exclusive or". It works the way "or" often does in English: "You can have your cake xor you can eat it."
`not`	The opposite: *true* becomes *false* and *false* becomes *true*. It works on just one value.

a	b	a and b	a or b	a xor b	not a
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Truth table: every combination of two Boolean values *a* and *b*, checked in BASIC-256 2.3.



NEW CONCEPT

```
expression1 and expression2
expression1 or expression2
expression1 xor expression2
not expression
```

The logical operators **and**, **or**, **xor** and **not** join or change comparisons. BASIC-256 works out the comparisons first, then **not**, then **and**, and **or** last. If you aren't sure, use brackets: `(age < 13 or age > 19) and hungry` says exactly what you mean.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 5

Blocks of statements — if, else and end if

When a decision needs more than one statement, put them in a block.

Often one statement isn't enough when a condition is true. Then you leave the statement off the **if** line, put as many statements as you need on the lines below it, and close the block with **end if**.



NEW CONCEPT

```
if condition then
   statements to run when true
end if
```

The statements between **if** … **then** and **end if** are executed only when the condition is true. Indent them, so that you can see at a glance where the block starts and ends. You may write **end if** or **endif**.



NEW CONCEPT

Edit ▸ **Beautify** tidies up your program: it removes extra spaces at the start and end of lines and indents blocks of code, like the statements inside **if** and **end if**.

A block can contain other **if** statements. In the next program the two inner **if**s are only checked when the outer one is true — when you rolled doubles.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

dice.kbs BASIC-256

```
# dice.kbs - roll two 6-sided dice
die1 = int(rand * 6) + 1
die2 = int(rand * 6) + 1
total = die1 + die2

print "die 1 = " + die1
print "die 2 = " + die2
message = "You rolled " + total + "."

if die1 = die2 then
   message += " Doubles."
   if total = 2 then
      message += " Snake eyes."
   end if
   if total = 12 then
      message += " Box cars."
   end if
end if

print message
```

TEXT OUTPUT

```
die 1 = 6
die 2 = 2
You rolled 8.
```

```
die 1 = 1
die 2 = 1
You rolled 2. Doubles. Snake eyes.
```

Program 40 · Rolling Dice

Deciding both ways — else

The last form of **if** adds **else**: one block runs when the condition is true and the other block runs when it is false. Exactly one of the two always runs.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



NEW CONCEPT

```
if condition then
   statements to run when true
else
   statements to run when false
end if
```

The block after **then** runs when the condition is *true*; the block after **else** runs when it is *false*. For a single statement on each side you can also write it on one line:

```
if coin < 0.5 then print "Heads." else print "Tails."
```

Program 41 rewrites the coin flip with **else**, so the second comparison isn't needed any more.

coinflip2.kbs BASIC-256

```
# coinflip2.kbs - coin flip with else
coin = rand
if coin < 0.5 then
   print "Heads."
   say "Heads."
else
   print "Tails."
   say "Tails."
end if
```

TEXT OUTPUT

```
Tails.
```

says **"Tails."**

Program 41 · Coin Flip — With Else

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



BIG PROGRAM

Roll a Die and Draw It

This chapter's big program rolls a single die and draws the dots on the graphics output — then tells you, out loud, what you rolled.

dieroll.kbs — BASIC-256

```
# dieroll.kbs - roll a 6-sided die on the screen

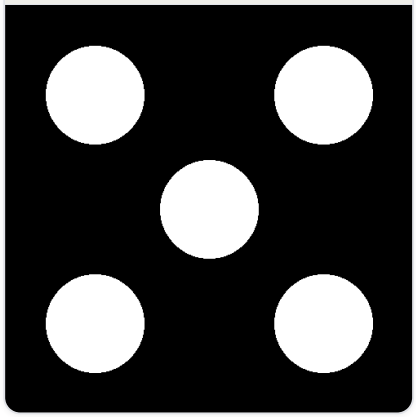
# radius of the dots
r = 60
# z1, z2 and z3 are the centres of the dots
# in each row and column
z1 = 110
z2 = 250
z3 = 390

# get roll
roll = int(rand * 6) + 1

clg black
color white
# top row
if roll <> 1 then circle z1,z1,r
if roll = 6 then circle z2,z1,r
if roll >= 4 and roll <= 6 then circle z3,z1,r
# middle row
if roll = 1 or roll = 3 or roll = 5 then circle z2,z2,r
# bottom row
if roll >= 4 and roll <= 6 then circle z1,z3,r
if roll = 6 then circle z2,z3,r
if roll <> 1 then circle z3,z3,r

message = "You rolled a " + roll + "."
print message
say message
```

GRAPHICS OUTPUT



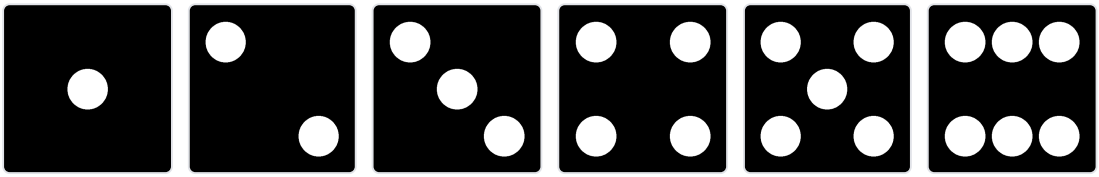
TEXT OUTPUT

```
You rolled a 5.
```

says **"You rolled a 5."**

Program 42 · Big Program — Roll a Die and Draw It

Each dot is drawn by its own **if** statement: the corner dots for every roll except 1, the middle dot only for the odd numbers, and the two middle dots of the top and bottom rows only for 6. These are all six faces the program can draw:



© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



PRACTICE

Exercises

WORD SEARCH

n	v	s	w	p	e	s	i	f	c	r	v	s
r	e	t	a	e	r	g	y	o	n	o	g	n
o	m	d	v	a	a	o	b	g	s	t	f	d
w	o	m	n	v	p	u	r	l	s	a	k	a
f	d	f	p	n	m	a	o	t	e	r	t	m
t	n	i	d	a	o	n	o	r	l	e	b	r
s	a	d	y	e	c	d	y	u	k	p	r	t
e	r	n	w	l	k	n	e	e	a	o	m	r
e	j	e	g	o	k	o	b	t	n	e	d	n
d	b	p	r	o	x	o	r	o	e	s	f	e
r	j	d	b	b	k	t	r	f	o	l	s	h
c	o	i	m	m	v	v	t	o	n	a	s	t
t	r	l	a	u	q	e	y	e	c	f	p	e

Find: and, boolean, compare, else, endif, equal, false, greater, if, less, not, operator, or, random, seed, then, true, xor



Problems

Try these — they only use what this chapter and the earlier ones covered.

1 Write a program that tosses a coin and tells you if your guess was right. Store a random number in a variable, then ask the user to type `h` or `t` (for heads or tails). If the number is less than 0.5 and the user typed `h`, or the number is 0.5 or more and the user typed `t`, tell them they won the toss.

2 Change your program from problem 1 so that it also tells the user when they did **not** win the toss. (Hint: **else**.)

3 Write a program that draws one round of rock, paper, scissors. Use two variables and give each one a random number. If a number is less than 1/3 it is rock, from 1/3 up to 2/3 it is paper, and 2/3 or more it is scissors. Show what both players drew.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

4 Add rules to your program from problem 3 to say who won. Remember: “paper covers rock”, “rock smashes scissors” and “scissors cut paper”. If both players drew the same thing, the round is a “draw”.

5 Add graphics and sound to your game from problem 4. Draw paper as a white rectangle, rock as a `darkorange` circle and scissors as a red X, and have the computer say who won.

6 In Program 42, `roll >= 4 and roll <= 6` could be shortened to `roll >= 4`. Why? Make the change and check that all six faces are still drawn correctly.

7 Add a `seed` statement at the top of Program 40 and try different seeds until you find one that rolls doubles. Run it again: does it still roll doubles?

Chapter 7 · Decisions, Decisions, Decisions. Modernised layout of the BASIC-256 beginner’s guide, updated for BASIC-256 2.3. Programs are numbered 37–42, continuing from Chapter 6, and a new Program 39 shows `seed`. New in this edition: string comparison puts capital letters before lower case ones, the order in which logical operators are worked out, the one-line `if … then … else`, and a look ahead to gaussian and OpenSimplex random numbers. Corrected: the `xor` truth table was labelled “OR”, and the Big Program is redrawn for the 500×500 graphics output. Every sample output was produced by running the program in BASIC-256 2.3. Original text © 2019 James M. Reneau, licensed under CC BY-NC-SA 3.0 US. This adaptation is shared under the same licence. Mascot: BitBot.