# CSCI 145 · Quiz 8 Reference Sheet (Lectures 15–16)

## BITS

- A bit is 0 (off) or 1 (on)
- n bits give $2^n$ patterns: the numbers 0 to $2^n - 1$
- Nibble = 4 bits, byte = 8 bits (0 to 255), word = 64 bits on most laptops

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^n$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |

## BINARY (BASE 2)

| Place | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|

- Binary to decimal: add the places that hold a 1
- Decimal to binary, two ways:
  - Subtract the biggest place that fits, write 1 there, repeat
  - Divide by 2: `num % 2` is the last bit, `num // 2` drops it; read the remainders bottom up
- A subscript gives the base when it isn't clear: $101_2 = 5$

**Adding:** 0 + 1 = 1, 1 + 1 = 10 (write 0, carry 1), 1 + 1 + 1 = 11 (write 1, carry 1). The sum can need one more bit

## HEXADECIMAL (BASE 16)

- Digits 0–9, then A–F for 10–15. One hex digit = 4 bits
- Places: ... 256, 16, 1. Two hex digits = one byte, `00` to FF = 0 to 255
- Hex to decimal: first digit × 16 + second digit (see the table's × 16 column)
- Decimal to hex: `num // 16` is the first digit, `num % 16` the second
- Binary ↔ hex: four bits at a time, from the right

| Dec | Binary | Hex | × 16 |
|---|---|---|---|
| 0 | 0000 | **0** | 0 |
| 1 | 0001 | **1** | 16 |
| 2 | 0010 | **2** | 32 |
| 3 | 0011 | **3** | 48 |
| 4 | 0100 | **4** | 64 |
| 5 | 0101 | **5** | 80 |
| 6 | 0110 | **6** | 96 |
| 7 | 0111 | **7** | 112 |

| Dec | Binary | Hex | × 16 |
|---|---|---|---|
| 8 | 1000 | **8** | 128 |
| 9 | 1001 | **9** | 144 |
| 10 | 1010 | **A** | 160 |
| 11 | 1011 | **B** | 176 |
| 12 | 1100 | **C** | 192 |
| 13 | 1101 | **D** | 208 |
| 14 | 1110 | **E** | 224 |
| 15 | 1111 | **F** | 240 |

## COLORS

- #RRGGBB: red, green, blue, one byte (two hex digits) each
- 3 bytes = 24 bits, so $2^{24}$ colors

## BINARY AND HEX IN PYTHON

| | |
|---|---|
| `0b1010` | A binary int literal (10) |
| `0x1a` | A hex int literal (26); either case |
| `bin(num)` | A **string** starting `'0b'`, no leading zeros |
| `hex(num)` | A **string** starting `'0x'`, lowercase digits |
| `int(text, 2)` | A binary string to an int |
| `int(text, 16)` | A hex string to an int; either case |

## NEGATIVE NUMBERS: TWO'S COMPLEMENT

- **Unsigned** (Lecture 15): every place is positive; 8 bits hold 0 to 255
- **Signed two's complement**: the leftmost place is negative; the rest are as before; 8 bits hold −128 to +127

| 8 bits | −128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|

- Every negative number starts with 1
- **Negate** a number: flip every bit, then add 1. The same two steps work in both directions
- Add as before; drop a carry out of the left end
- **Overflow**: the answer needs more bits than there are

| # of Bits | Smallest | Largest |
|---|---|---|
| 4 | −8 | 7 |
| 8 | −128 | 127 |
| n | $-2^{n-1}$ | $2^{n-1} - 1$ |

Python ints grow as big as they need to

## FLOATS ARE APPROXIMATE

- Python stores real numbers as 64-bit floats
- Ints are exact; floats are not. Many values, like 0.1, can't be stored exactly: there are gaps between floats
- Sums of halves, quarters, eighths, ... are exact
- Don't compare floats with ==; check that they're close: `abs(a - b) < 0.000001`
- To stop at zero, test `f <= 0`, not `f == 0`: rounding can land just past 0
- Round before you show a float: `round(x, 2)`
- A float's exponent moves the point, so floats go far beyond $2^{64}$; big floats have big gaps

## TEXT

- Each character is stored as a number, its **code**
- ASCII: 128 codes (7 bits). Unicode: every language plus emoji; its first 128 codes match ASCII. UTF-8 stores a code in 1 to 4 bytes
- Uppercase and lowercase letters are 32 apart; letters and digits are in order
- Strings compare by code

| | |
|---|---|
| `ord(ch)` | The code of one character (an int) |
| `chr(code)` | The character with that code (a string) |

## ASCII

Row = first hex digit, column = second. Decimal code = the row's × 16 + the column

| | ×16 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 32 | sp | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3 | 48 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | 64 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5 | 80 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6 | 96 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7 | 112 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | del |

Column header letters are hex digits: A = 10, ..., F = 15. sp = space. Codes 0–31 are invisible; newline is 10