

Input 5: Time, Date

This unit introduces using the current time and date as inputs.

Syntax introduced:

`second()`, `minute()`, `hour()`, `millis()`, `day()`, `month()`, `year()`

Humans have a relative perception of time, but machines attempt to keep precise, regular time. Previous civilizations used sundials and water clocks to visualize the passage of time; today, most people use the digital numeric clock and the twelve-hour circular clock with a minute, second, and hour hand. Each of these representations reflects their technology. A numeric digital readout is appropriate for a digital timekeeping mechanism in need of an inexpensive display. A timekeeping mechanism built from circular gears lends itself to a circular presentation of time. The tower-sized clocks of the past with enormous gears and weights have evolved and shrunk into devices so inexpensive and abundant that digital devices such as microwave ovens, mobile phones, and computers all display the time and date.

Reading the current time and date into a program opens an interesting area of exploration. Knowledge of the current time makes it possible to write a program that changes its colors every day depending on the date, or a digital clock that plays an animation every hour. The ability to input time and date information enables software tools for remembering, reminding, and informing and creates the potential for whimsical time-based events.

Seconds, Minutes, Hours

Processing programs can read the value of the computer's clock. The current second is read with the `second()` function, which returns values from 0 to 59. The current minute is read with the `minute()` function, which also returns values from 0 to 59. The current hour is read with the `hour()` function, which returns values in the 24-hour time notation from 0 to 23. In this system midnight is 0, noon is 12, 9:00 a.m. is 9, and 5:00 p.m. is 17. Run this program to see the current time:

```
int s = second(); // Returns values from 0 to 59
int m = minute(); // Returns values from 0 to 59
int h = hour();   // Returns values from 0 to 23
println(h + ":" + m + ":" + s); // Prints the time to the console
```

28-01

Placing these functions inside `draw()` allows the time to be read continuously. This example reads the current time and updates the text area with the passage of each second:

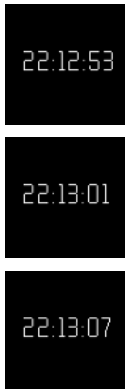
```
int lastSecond = 0;
```

28-02

```
void setup() {
  size(100, 100);
}

void draw() {
  int s = second();
  int m = minute();
  int h = hour();
  // Only prints once when the second changes
  if (s != lastSecond) {
    println(h + ":" + m + ":" + s);
    lastSecond = s;
  }
}
```

You can create a clock with a numerical display by using the `text()` function to draw the numbers to the display window. The `nf()` function (p. 422) is used to space the numbers equally from left to right. Single digit numbers are padded on their left with a zero so all numbers occupy a two-digit space at all times.



```
PFont font;

void setup() {
  size(100, 100);
  font = loadFont("Pro-20.vlw");
  textFont(font);
}

void draw() {
  background(0);
  int s = second();
  int m = minute();
  int h = hour();
  // The nf() function spaces the numbers nicely
  String t = nf(h,2) + ":" + nf(m,2) + ":" + nf(s,2);
  text(t, 10, 55);
}
```

28-03

There are many different ways to express the passage of time. In the next example, horizontal lines mark the current second, hour, and minute. The left edge of the display window is 0 and the right edge is the maximum for each time component. Because the values from the time functions range from 0 to 59 and from 0 to 23, they are modified to

all have the range of 0 to 99. Each time value is divided by its maximum value and then multiplied by 100.0.



```
void setup() {
  size(100, 100);
  stroke(255);
}
```

28-04



```
void draw() {
  background(0);
  float s = map(second(), 0, 60, 0, 100);
  float m = map(minute(), 0, 60, 0, 100);
  float h = map(hour(), 0, 24, 0, 100);
  line(s, 0, s, 33);
  line(m, 34, m, 66);
  line(h, 67, h, 100);
}
```



These normalized time values can also be used to simulate the second, minute, and hour hands on a traditional clock face. In this case, the values are multiplied by 2π to display the time as points around a circle. Because the `hour()` function returns values in 24-hour time, the program converts the hour value to 12-hour time scale by calculating the `hour % 12` (the `%` symbol is introduced on p. 45).



```
void setup() {
  size(100, 100);
  stroke(255);
}
```

28-05



```
void draw() {
  background(0);
  fill(80);
  noStroke();
  // Angles for sin() and cos() start at 3 o'clock;
  // subtract HALF_PI to make them start at the top
  ellipse(50, 50, 80, 80);
  float s = map(second(), 0, 60, 0, TWO_PI) - HALF_PI;
  float m = map(minute(), 0, 60, 0, TWO_PI) - HALF_PI;
  float h = map(hour() % 12, 0, 12, 0, TWO_PI) - HALF_PI;
  stroke(255);
  line(50, 50, cos(s) * 38 + 50, sin(s) * 38 + 50);
  line(50, 50, cos(m) * 30 + 50, sin(m) * 30 + 50);
  line(50, 50, cos(h) * 25 + 50, sin(h) * 25 + 50);
}
```



In addition to reading the current time, each Processing program counts the time passed since the program started. This time is stored in milliseconds (thousandths of a second). Two thousand milliseconds is 2 seconds, and 200 milliseconds is 0.2 seconds. This number is obtained with the `millis()` function and can be used to trigger events and calculate the passage of time:

```
// Uses millis() to start a line in motion three seconds  
// after the program starts
```

28-06

```
int x = 0;  
  
void setup() {  
    size(100, 100);  
}  
  
void draw() {  
    if (millis() > 3000) {  
        x++;  
    }  
    line(x, 0, x, 100);  
}
```

The `millis()` function returns an `int`, but it is sometimes useful to convert it to a `float` that represents the number of seconds elapsed since the program started. The resulting number can be used to control the sequence of events in an animation.

```
int x = 0;  
  
void setup() {  
    size(100, 100);  
}  
  
void draw() {  
    float sec = millis() / 1000.0;  
    if (sec > 3.0) {  
        x++;  
    }  
    line(x, 0, x, 100);  
}
```

28-07

Date

Date information is read in a similar way as the time. The current day is read with the `day()` function, which returns values from 1 to 31. The current month is read with the `month()` function, which returns values from 1 to 12 where 1 is January, 6 is June, and 12 is December. The current year is read with the `year()` function, which returns the four-digit integer value of the present year. Run this program to see the current date in the console:

```
int d = day();    // Returns values from 1 to 31
int m = month();  // Returns values from 1 to 12
int y = year();   // Returns four-digit year (2007, 2008, etc.)
println(d + " " + m + " " + y);
```

28-08

The following example checks to see if it is the first day of the month and prints the message “Welcome to a new month.” to the console if it is the first day of the month.

```
void draw() {
    int d = day(); // Values from 1 to 31
    if (d == 1) {
        println("Welcome to a new month.");
    }
}
```

28-09

The following example runs continuously and checks to see if it is New Year’s Day. If so, the message “Today is the first day of the year!” is printed to the console.

```
void draw() {
    int d = day(); // Values from 1 to 31
    int m = month(); // Values from 1 to 12
    if ((d == 1) && (m == 1)) {
        println("Today is the first day of the year!");
    }
}
```

28-10

Exercises

1. Make a simple clock to run an animation for two seconds at the beginning of each minute.
2. Create an abstract clock that communicates the passage of time through graphical quantity rather than numerical symbols.
3. Write a program to draw images to the display window corresponding to specific dates (e.g., display a pumpkin on Halloween).