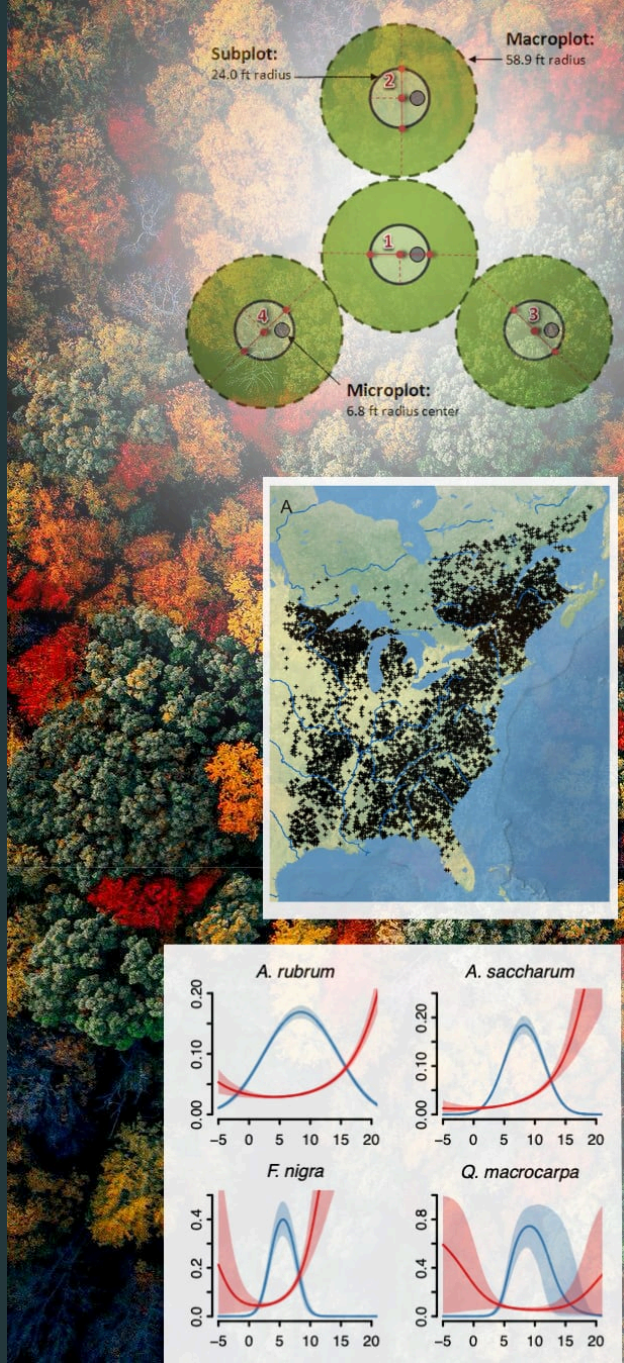
Lecture 3

Programming in R

Introduction to R for Biologists - Lauren Talluto



Looping

In programming, **loops** allow us to do something repeatedly, without copy-pasting code. One simple loop construct is called a `for` loop. Here, we create a **loop variable** `v` that takes the values from an **iteration variable** (normally a vector) named `vec`.

The code between the `{}` symbols is called a **block**.

```
vec = c(1, 5, 3, 6, 8)

for(v in vec) {
    print(v)
}
## [1] 1
## [1] 5
## [1] 3
## [1] 6
## [1] 8
```

For loops — index format

Sometimes we want to iterate over something by **index** (for example, rows of a data frame, or using relative positions for some reason).

Here we compute the first 20 Fibbonaci numbers.

```
fib = numeric(20)
fib[1] = 0
fib[2] = 1

# start on 3, because the first two are already defined!
for(i in 3:length(fib)) {
    fib[i] = fib[i - 1] + fib[i - 2]
}
print(fib)
##  [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
## [16]  610  987 1597 2584 4181
```

Conditionals

A **conditional** statement makes a decision based on the value of an expression.

```
peng = read.csv("data/penguins.csv")
(body_masses = tapply(peng$body_mass_g, peng$species, mean, na.rm = TRUE))
##    Adelie Chinstrap    Gentoo
##  3700.662  3733.088  5076.016
```

```
if(body_masses['Gentoo'] > body_masses['Chinstrap']) {
    print("Gentoo penguins are larger than Chinstrap penguins")
}
## [1] "Gentoo penguins are larger than Chinstrap penguins"
```

Conditionals — else clauses

An **else clause** allows you to make a binary choice

```
(bill_length = tapply(peng$bill_length_mm, peng$species, mean, na.rm = TRUE))
##    Adelie Chinstrap    Gentoo
##  38.79139  48.83382  47.50488
```

```
if(bill_length['Gentoo'] > bill_length['Chinstrap']) {
    print("Gentoo penguins have larger bills than Chinstrap penguins")
} else {
    print("Chinstrap penguins have larger bills than Gentoo penguins")
}
## [1] "Chinstrap penguins have larger bills than Gentoo penguins"
```

Conditionals — chaining clauses

You can chain together multiple either-or conditionals with **else if**

```
biggest_bill = max(bill_length)
if(bill_length['Gentoo'] == biggest_bill) {
    print("Gentoo penguins have the longest bills")
} else if(bill_length['Chinstrap'] == biggest_bill) {
    print("Chinstrap penguins have the longest bills")
} else {
    print("Adelie penguins have the longest bills")
}
## [1] "Chinstrap penguins have the longest bills"
```

Functions

You can write your own functions using the `function` keyword.

Here we write a small function to decide if an input value is a prime number, up to a maximum of 1000.

```
is_prime = function(x) {
    if(x > 1000)
        return(NA)

     # x cannot have factors larger than sqrt(x)
    max_factor = as(sqrt(x), "integer")
    result = TRUE # if we don't find a factor, the number is prime
    for(i in 2:max_factor) {
        # if x divides into any number with no remainder it is not prime
        if(x %% i == 0) {
            result = FALSE
        }
    }
    return(result)
}
```

Functions — default parameters

You can add defaults to parameters when you define a function.

Here we allow the user to decide the maximum, with a default of 1000.

```
is_prime = function(x, max_value = 1000) {
    if(x > max_value)
        return(NA)

     # x cannot have factors larger than sqrt(x)
    max_factor = as(sqrt(x), "integer")
    result = TRUE # if we don't find a factor, the number is prime
    for(i in 2:max_factor) {
        # if x divides into any number with no remainder it is not prime
        if(x %% i == 0) {
            result = FALSE
        }
    }
    return(result)
}
```

Vectorising is_prime

Here is some additional code we worked on in class.

```
is_prime = function(x, max_value = 1000) {
    if(any(x > max_value))
        return(NA)
    # x cannot have factors larger than sqrt(x)
    result = rep(TRUE, length(x)) # if we don't find a factor, the number is prime
    for(j in 1:length(x)) {
        max_factor = as(sqrt(x[j]), "integer")
        for(i in 2:max_factor) {
            # if x divides into any number with no remainder it is not prime
            if(x[j] %% i == 0) {
                result[j] = FALSE
            }
        }
    }
    return(result)
}

numbers = 1:50

is_prime(numbers)
ifelse(is_prime(numbers), "prime", "not prime")
```

Simulations and Random Numbers

Many times we need random numbers. Here are some useful functions for doing this.

```
# x is a vector, chooses n random items from x
sample(x, n)

# take n random numbers between min and max
runif(n, min, max)

# take n random integers from a poisson distribution
# rate gives the average result
rpois(n, rate)

# take n random numbers from a normal distribution
rnorm(n, mean, sd)
```

Random numbers: example

```
card_suits = c('hearts', 'diamonds', 'clubs', 'spades')
card_vals = c(as.character(2:10), 'J', 'Q', 'K', 'A')
card_deck = paste(rep(card_vals, 4), rep(card_suits, each = 13), sep = '-')
head(card_deck)
## [1] "2-hearts" "3-hearts" "4-hearts" "5-hearts" "6-hearts" "7-hearts"
```

```
(poker_hand = sample(card_deck, 5, replace = FALSE))
## [1] "6-diamonds" "10-clubs"   "6-hearts"   "8-hearts"   "A-diamonds"
```