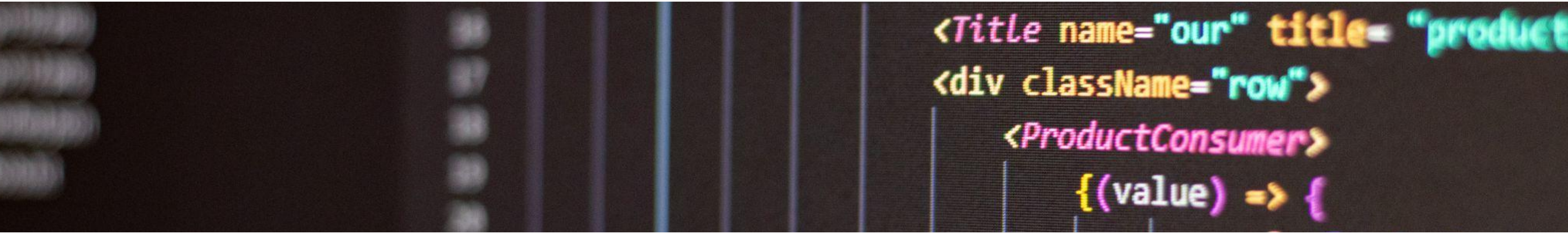



# Scientific Programming: The Use of Loops

**Robert Ladwig, PhD**

Environmental Systems Engineering, teaching seminar at SUNY ESF

ladwigjena@gmail.com

@hydrobert

robertladwig.github.io

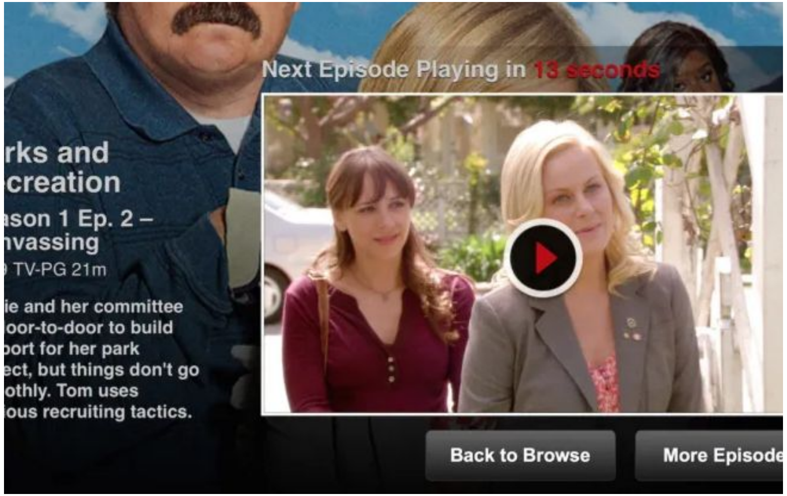
robertladwig

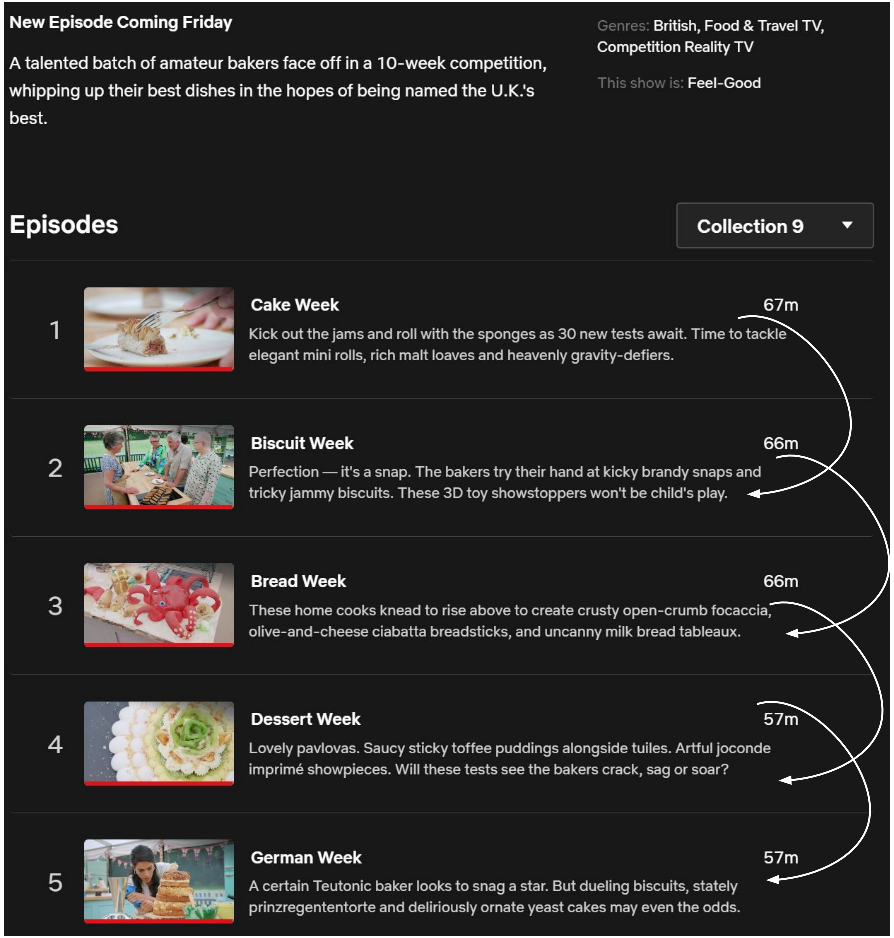
# Daily life loops?

# Daily life loops?

Netflix's 'post-play' feature:

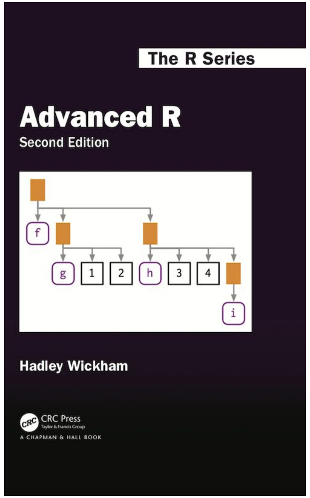
- play every episode of that season after another

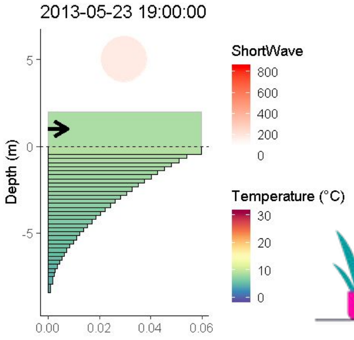


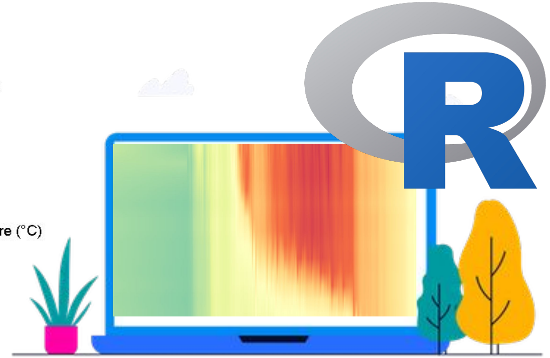



# Expectations on today's topic

- important concept of control flow in scientific programming
- example code is in R (*Advanced R*, https://adv-r.hadley.nz/)
- review of the use of loops
- application for water quality modeling
- **material**: https://github.com/robertladwig/1DDiffusionExample

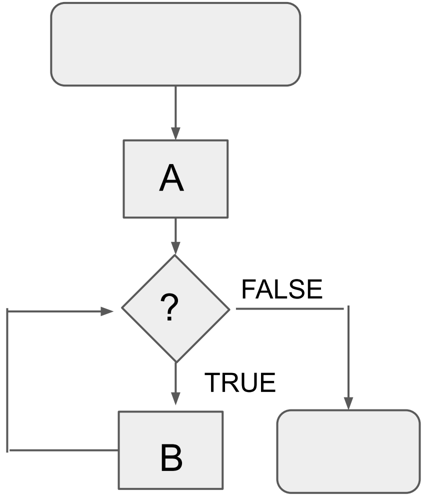


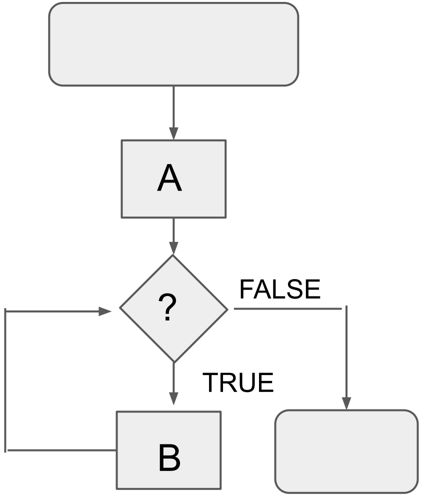
# The Control Flow concept

- **control flow**: order in which statements and calls are executed
- can have pathways and options



# The Control Flow concept

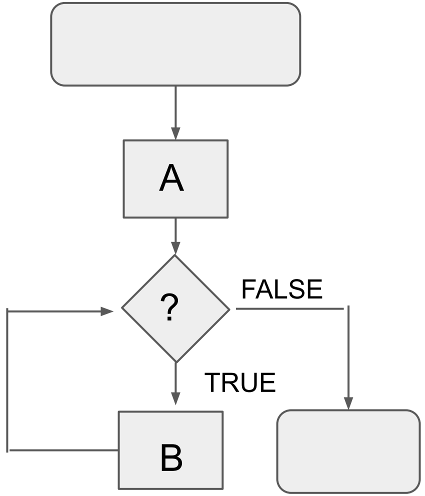
- **control flow**: order in which statements and calls are executed
- can have pathways and options
- (mostly) **choices** or **loops**
  - choices are `if` statements or `switch()`




```
> if(b > a){
     c = b - a
}
> print(c)
[1] 1
```

```
> lecture <- switch("Topic",
"Teacher" = "Robert Ladwig",
"Topic" = "Use of Loops")

> print(lecture)

[1] "Use of Loops"
```

# The Control Flow concept




- **control flow**: order in which statements and calls are executed
- can have pathways and options
- (mostly) **choices** or **loops**
  - choices are `if` statements or `switch`
- **today's topic: loops**

```
> if(b > a){
    c = b - a
}
> print(c)
[1] 1
```

```
> lecture <- switch("Topic",
"Teacher" = "Robert Ladwig",
"Topic" = "Use of Loops")

> print(lecture)

[1] "Use of Loops"
```

# Loops

- sequence of statements
- carried out iteratively and several times

$$\sum_{i=1}^{n=5} i = ?$$

# Loops

- sequence of statements
- carried out iteratively and several times

$$\sum_{i=1}^{n=5} i = 1 + 2 + 3 + 4 + 5 = 15$$

# Loops

- `for` or `while`

```
for (item in vector){

    perform_action

}
```

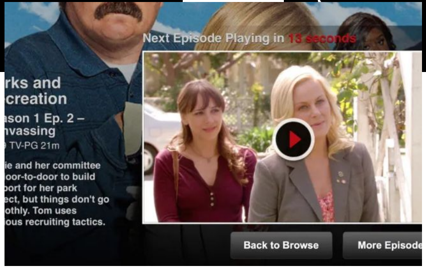
```
while (condition){

    perform_action

}
```

# Loops

- `for` or `while`

```
for ( Episode in Show ){
    perfo Watch tion
}
```

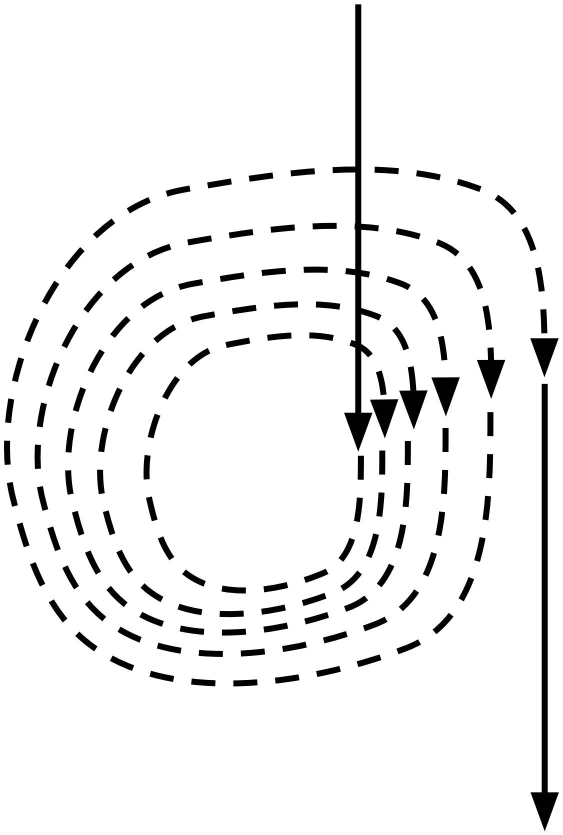
```
while ( There are unwatched episodes {
    perfo Watch tion
}
```



# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

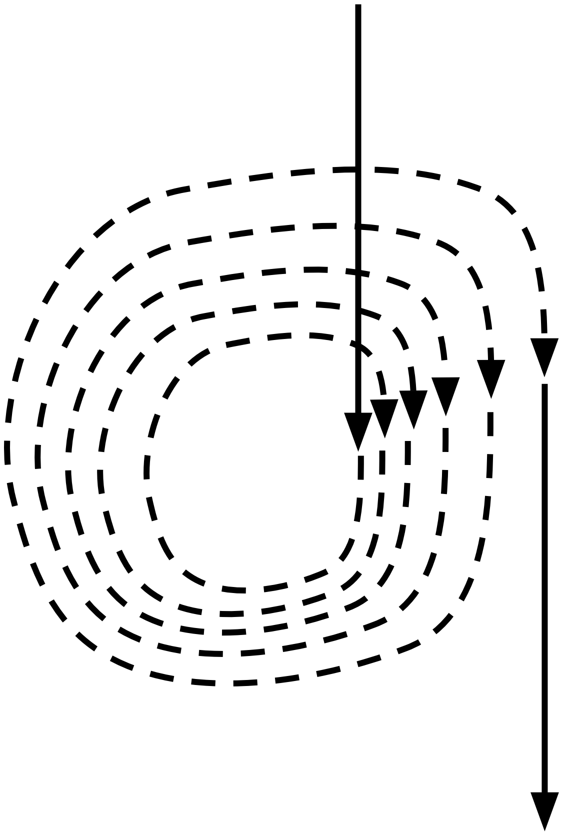


```
> i = 1
> n = 5
> x = 0
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

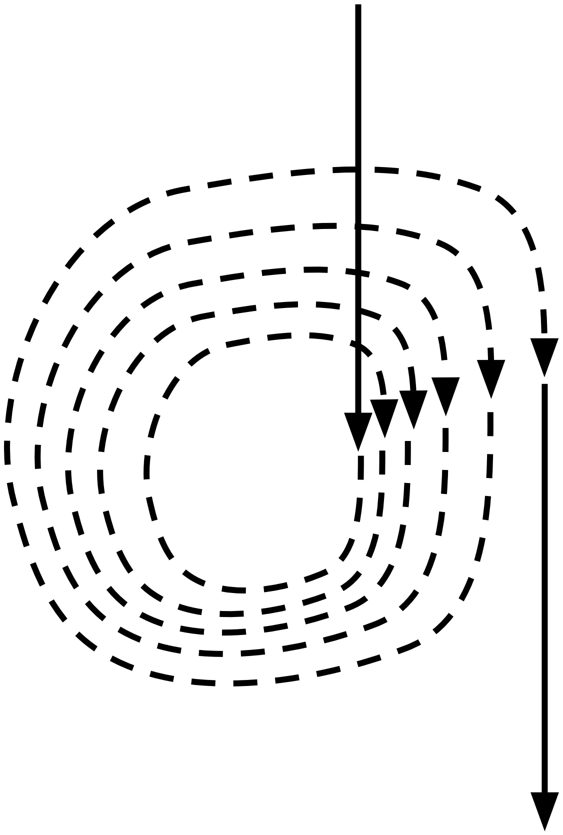


```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

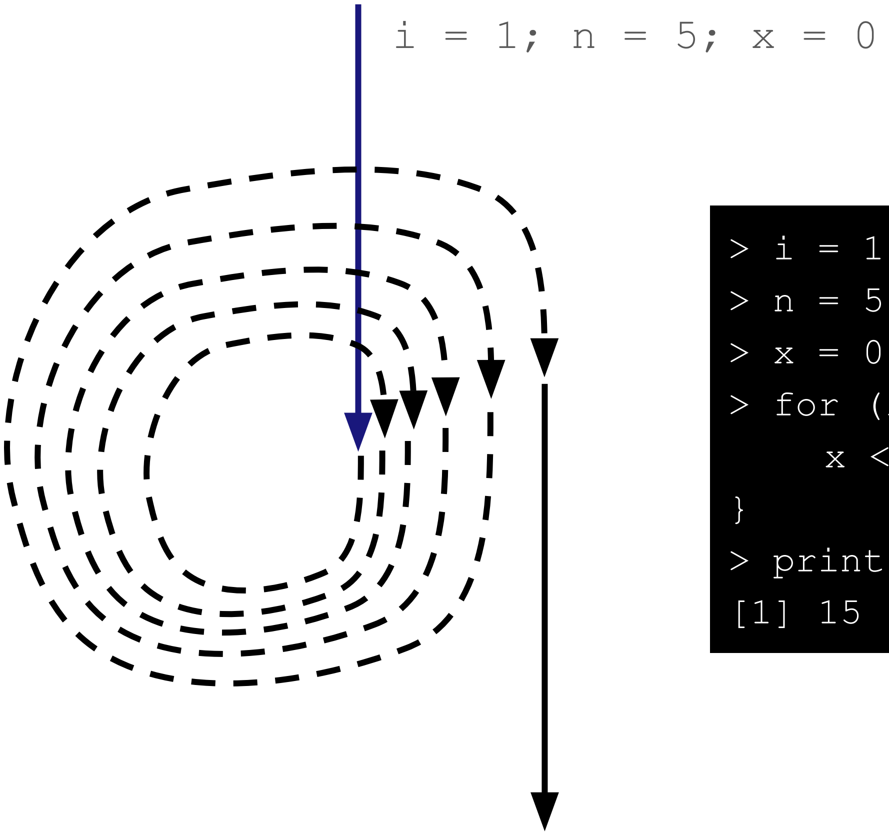


```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

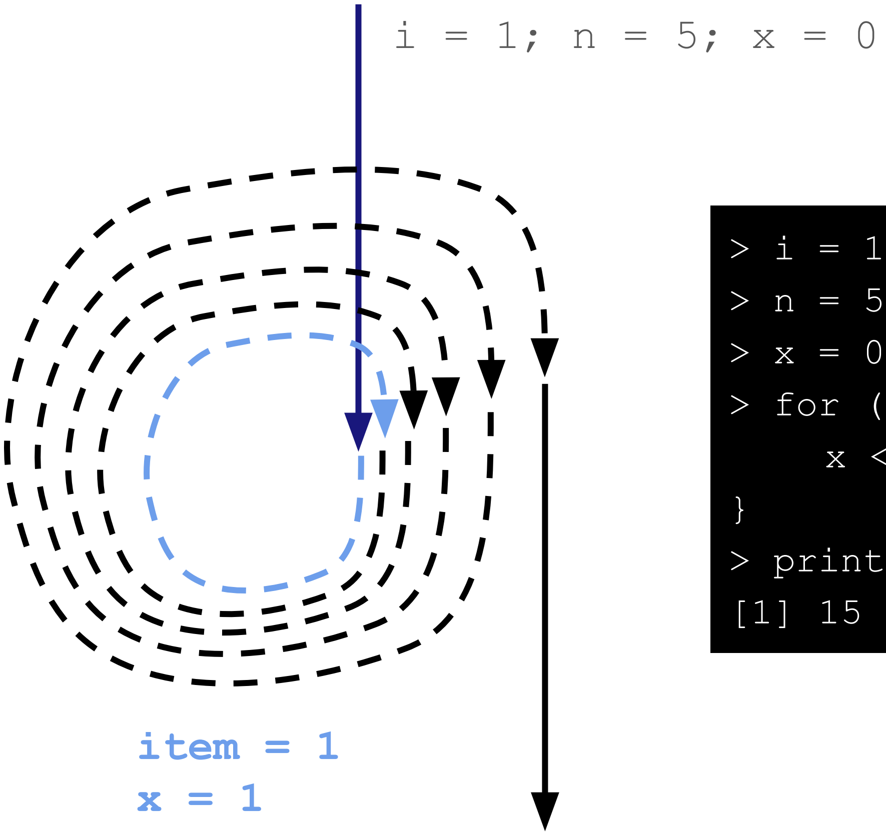



```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

# Loops

- for-loop

$$\sum_{i=1}^{n=5} i$$

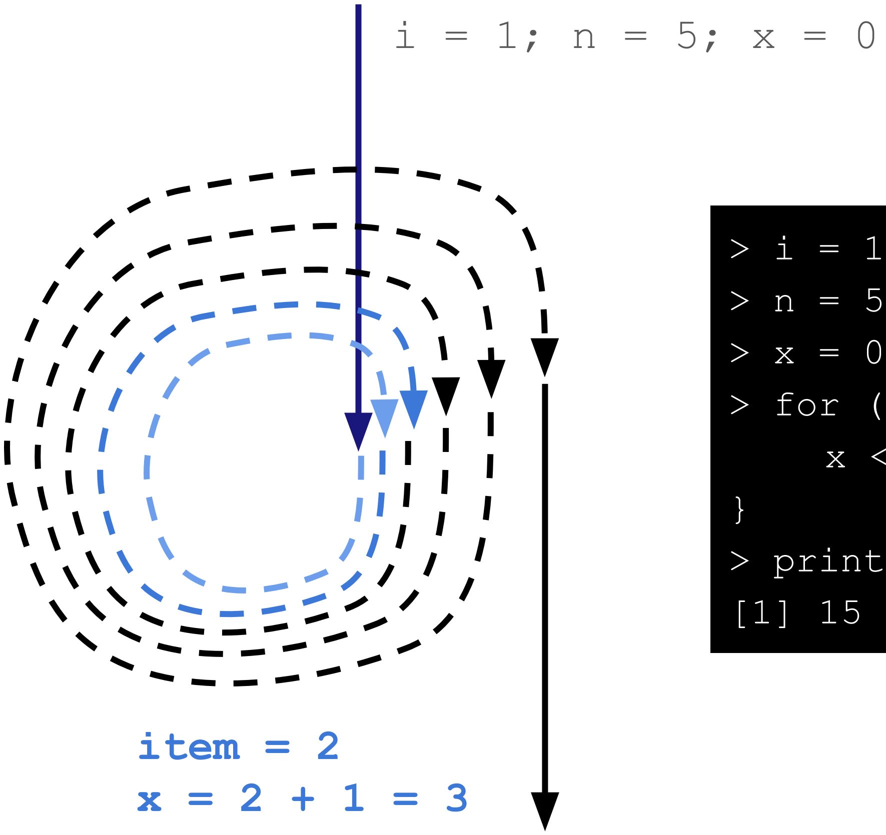



```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

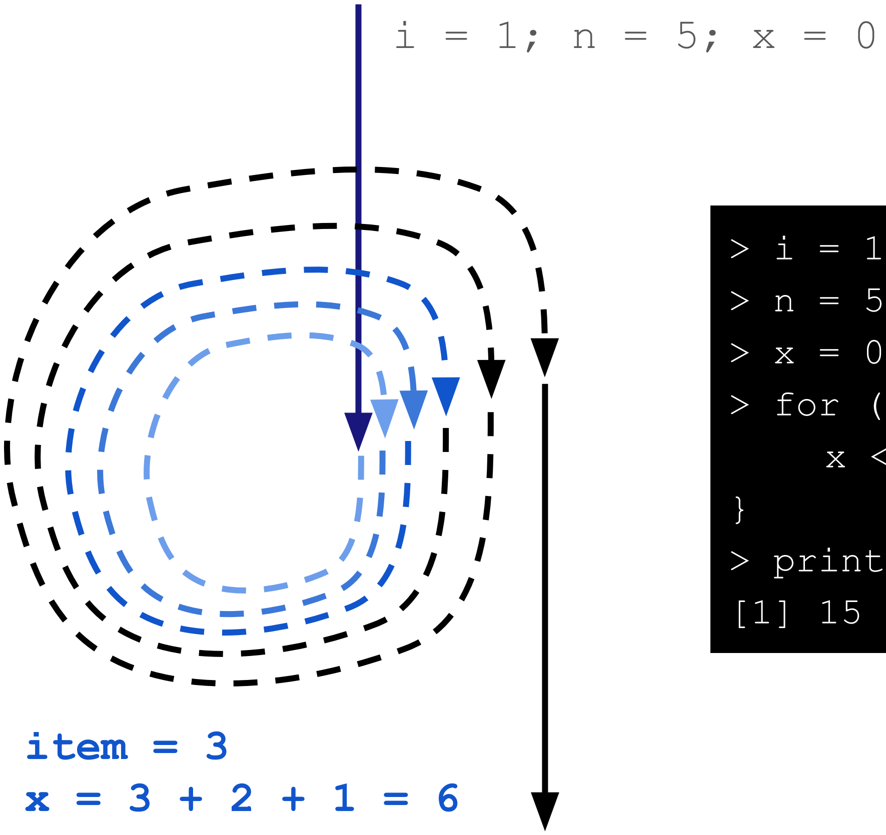



```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

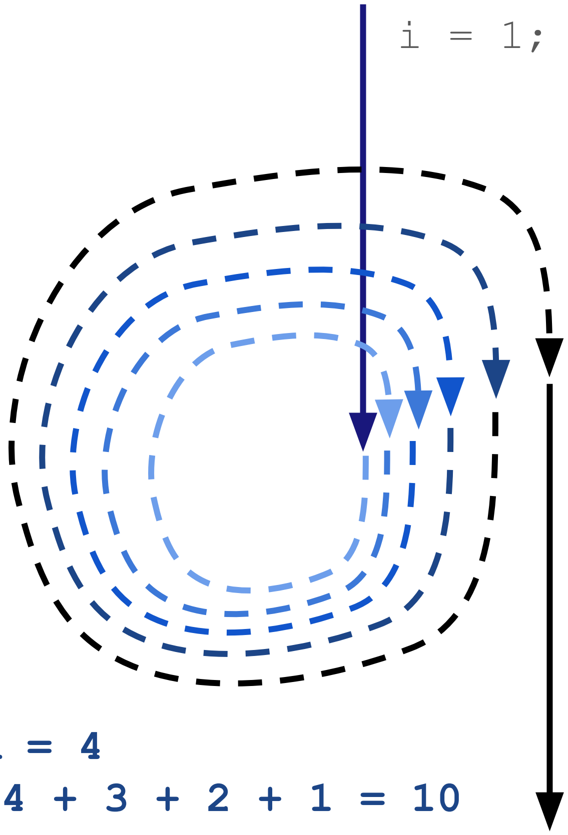



```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
      x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$



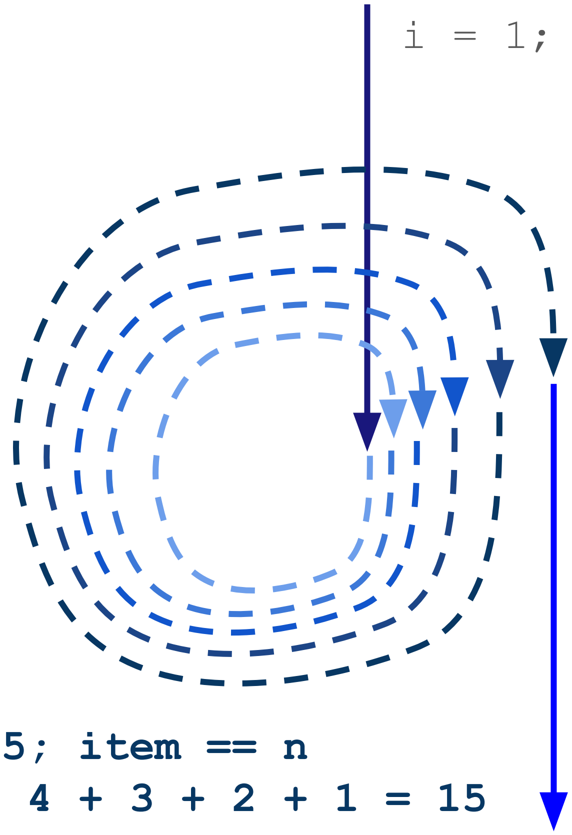

```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

`i = 1; n = 5; x = 0`



**`item = 5; item == n`**

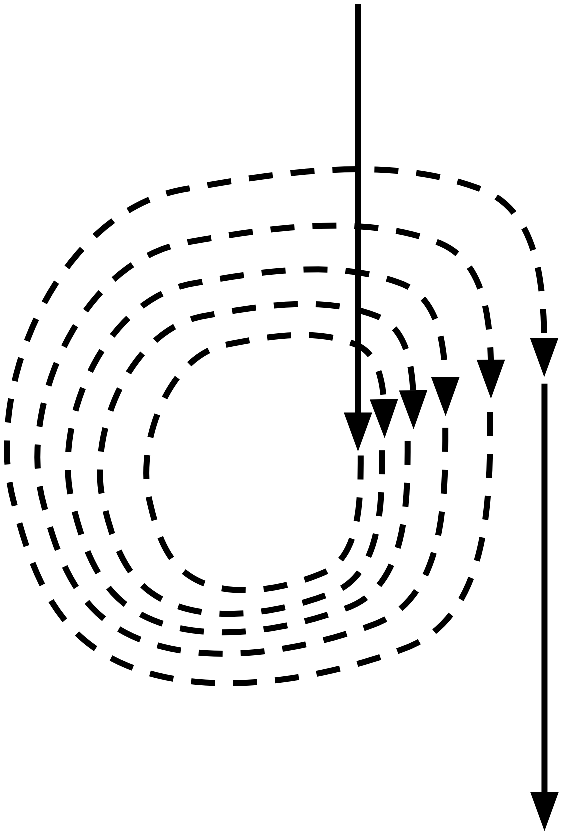
**`x = 5 + 4 + 3 + 2 + 1 = 15`**

```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
    x <- x + item
}
> print(x)
[1] 15
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

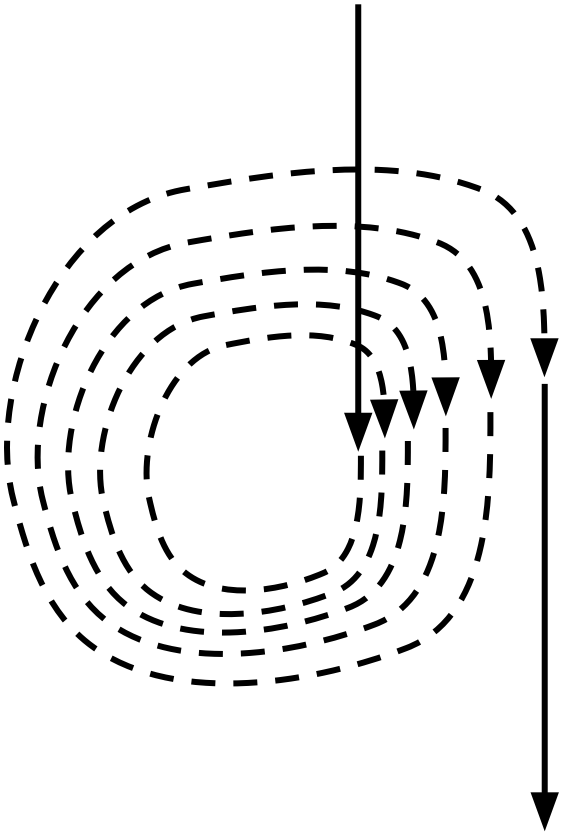


```
> numbers = [1, 2, 3, 4, 5]
> x = 0
> for item in numbers:
      x += item
> x
```

# Loops

- `for`-loop

$$\sum_{i=1}^{n=5} i$$

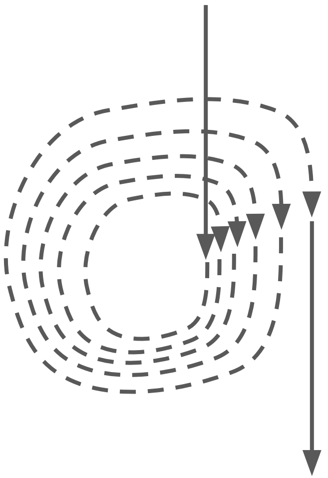


C

```
> int sum=0, number=5;
> for(int
i=1;i<=number;i++)
     {
       sum = sum + i;
     }
```

# Question: Loops

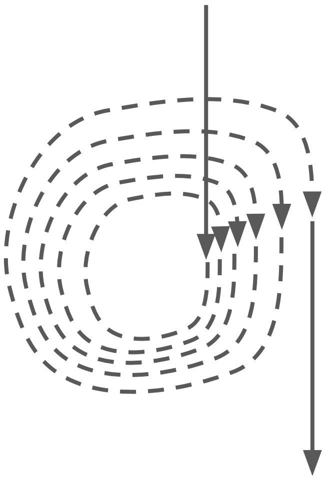
- `for`-loop



```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector, item * 2)
}
> vector
```

# Question: Loops

- `for`-loop

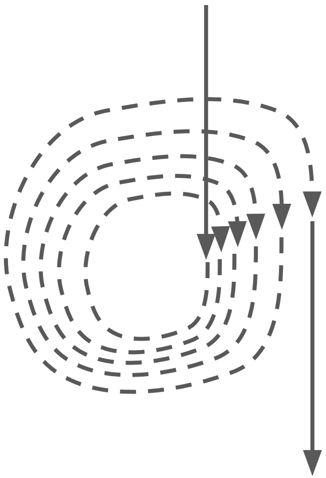


```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector, item * 2)
}
> vector
```

```
item = 1
vector <- c(c(1, 2, 3), 1 * 2)
```

# Question: Loops

- `for`-loop

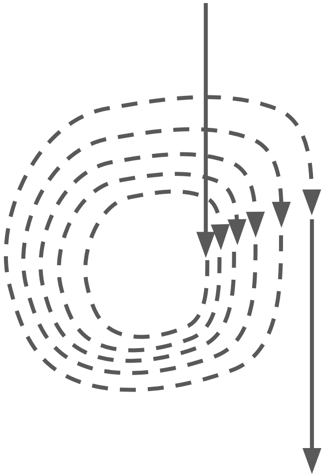


```
> vector <- c(1, 2, 3)
> for (item in vector) {
       vector <- c(vector, item * 2)
}
> vector
```

```
item = 2
vector <- c(c(1, 2, 3, 2), 2 * 2)
```

# Question: Loops

- `for`-loop

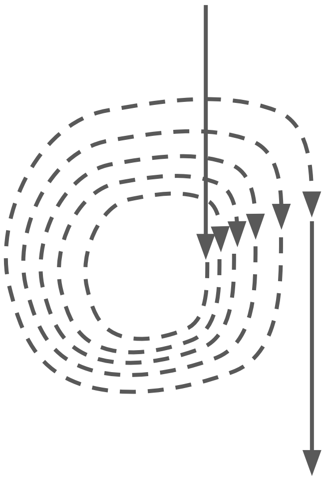


```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector, item * 2)
}
> vector
```

```
item = 3
vector <- c(c(1, 2, 3, 2, 4), 3 * 2)
```

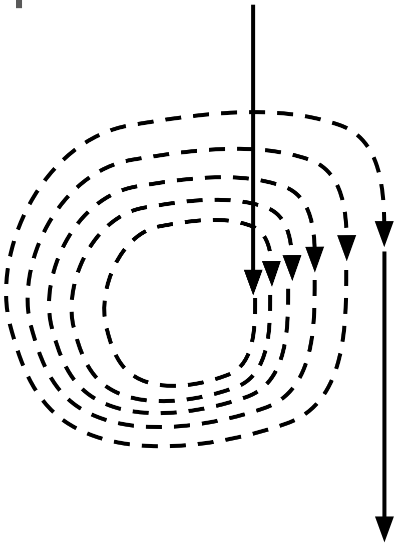
# Question: Loops

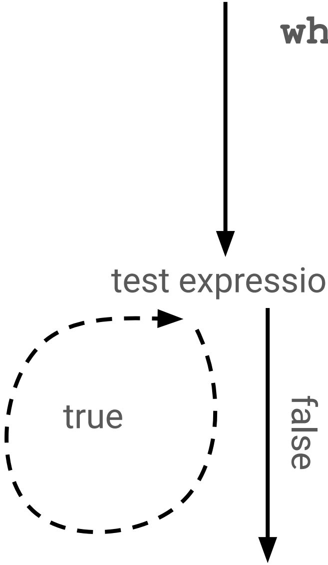
- `for`-loop



```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector, item * 2)
}
> vector
[1] 1 2 3 2 4 6
```

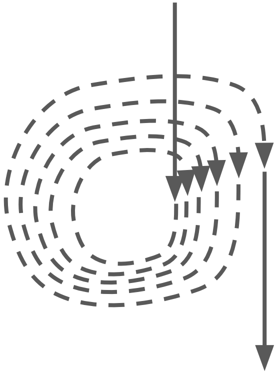
# Loops


`for`-loop
do it *n*-times


`while`-loop
check every time if test is valid
test expression
true
false

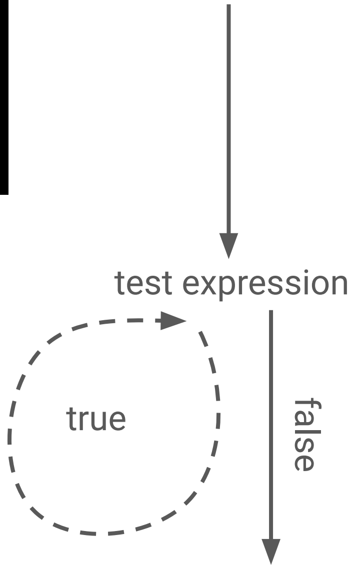
# Loops

- `while`-loop



```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
     x <- x + item
}
> print(x)
[1] 15
```

```
> item = 1
> n = 5
> x = 0
> while(item <= n){
     x <- x + item
     item = item + 1
}
> print(x)
[1] 15
```

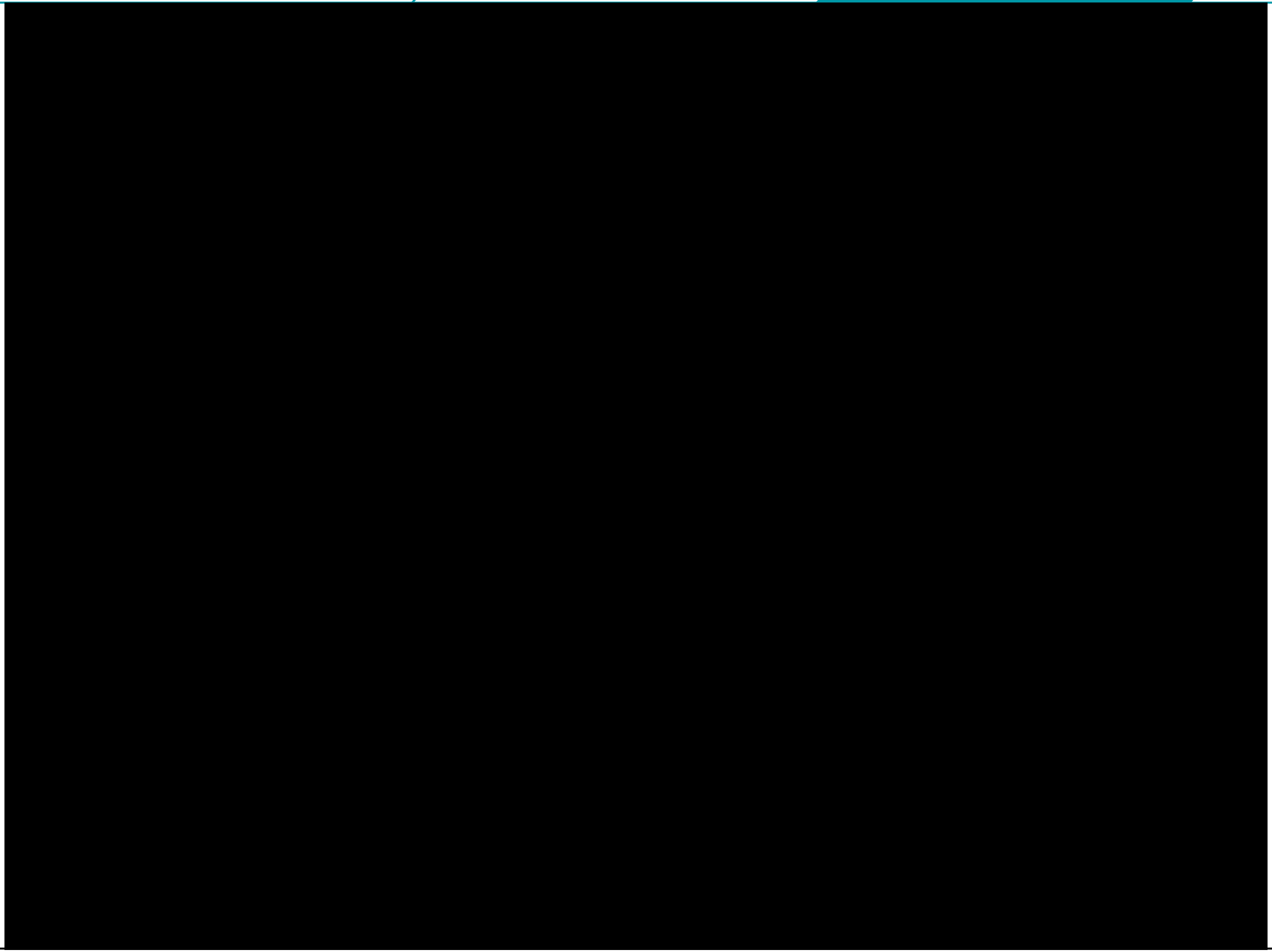


# Combining loops and choices

- the power of control flow statements: **finding prime numbers**
- natural numbers >1 and divisible by only one and the number itself

- %%: modulo operator, gives remainder of division

```
primeNum <- c()

for (item in 2:1000){
    prime <- TRUE

    i = 2
    while(i < item) {
        if(item %% i == 0){
            prime <- FALSE
        }
        i <- i + 1
    }

    if (prime){
        primeNum <- c(primeNum,
    item)
    }
}
```



Video by Engin Akyurt

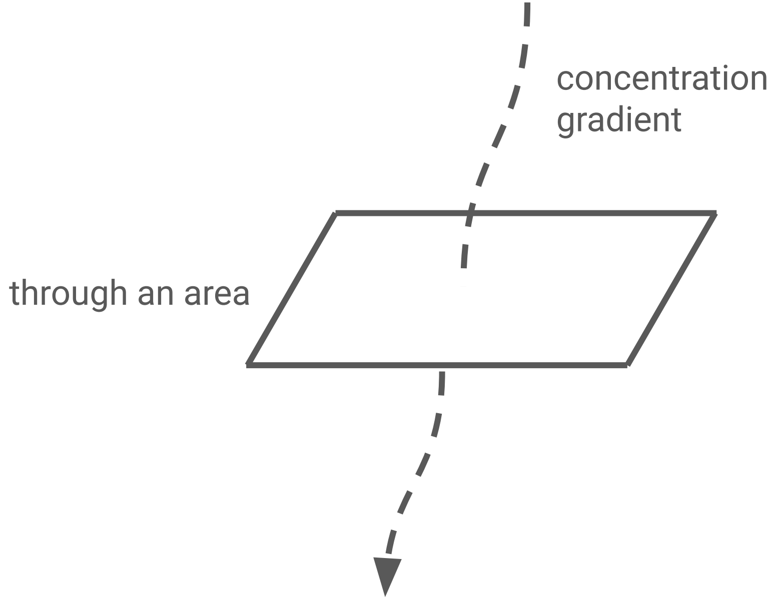
# Applying loops for water quality modeling

- in aquatic systems, **diffusion** is one of the main transport processes
  - **molecular diffusion**: random Brownian motion
  - **turbulent diffusion**: large scale motion by eddies
- motion by a diffusion coefficient and a (velocity/concentration) gradient

- **let's model the diffusion of a (passive) nutrient in a lake**
  - passive: no reaction/transformation

# Applying loops for water quality modeling

- code a model over **time and space**
- transport of a concentration over time




# Applying loops for water quality modeling

- code a model over **time and space**

diffusion coefficient

$$\frac{dC}{dt} = K \frac{d^2 C}{dz^2}$$

change of concentration over time

second-order derivative of the concentration over the vertical axis z

# Applying loops for water quality modeling

- code a model over **time and space**
- **check the units!**

area over time

$$\frac{dC}{dt} = K \frac{d^2 C}{dz^2}$$

mass over volume and time

mass over volume and area

$$\frac{g}{m^3 s} = \frac{m^2}{s} \frac{g}{m^3 m^2}$$

# Applying loops for water quality modeling

- luckily, we can discretize this using a **Central Difference Scheme**
- use of Taylor expansion (series expansion of a function)

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + O(a^4)$$

- centered differencing in space, forward differencing in time

# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**

- the power of loops!

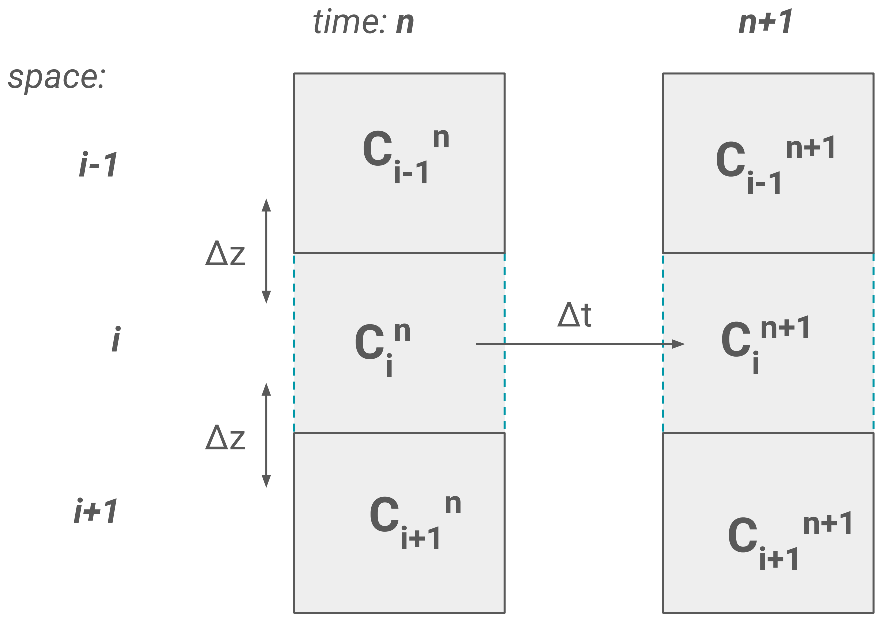
$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$

# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**

- the power of loops!

$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$

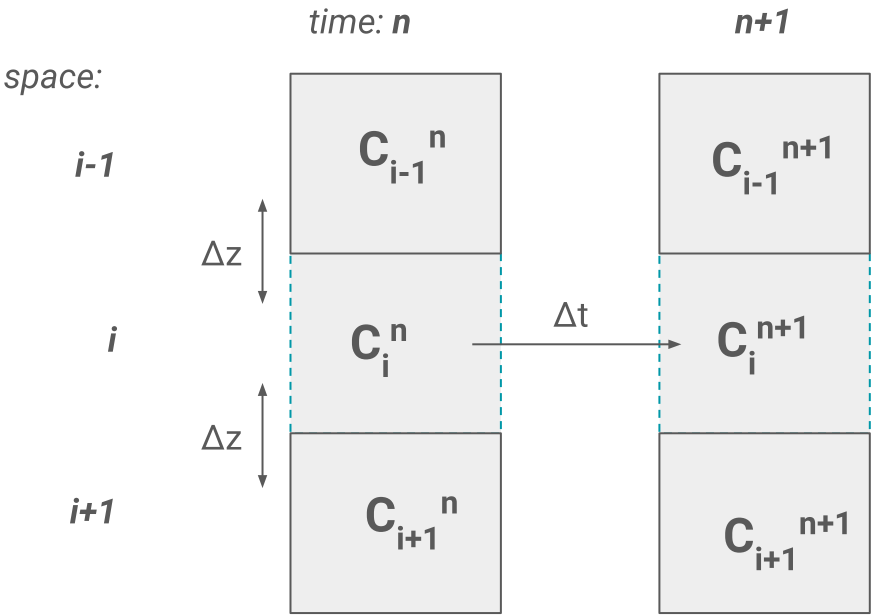


# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**

- the power of loops!

$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$




*Beware*!
this is an **explicit** scheme, therefore its numerical stability depends on the time step size

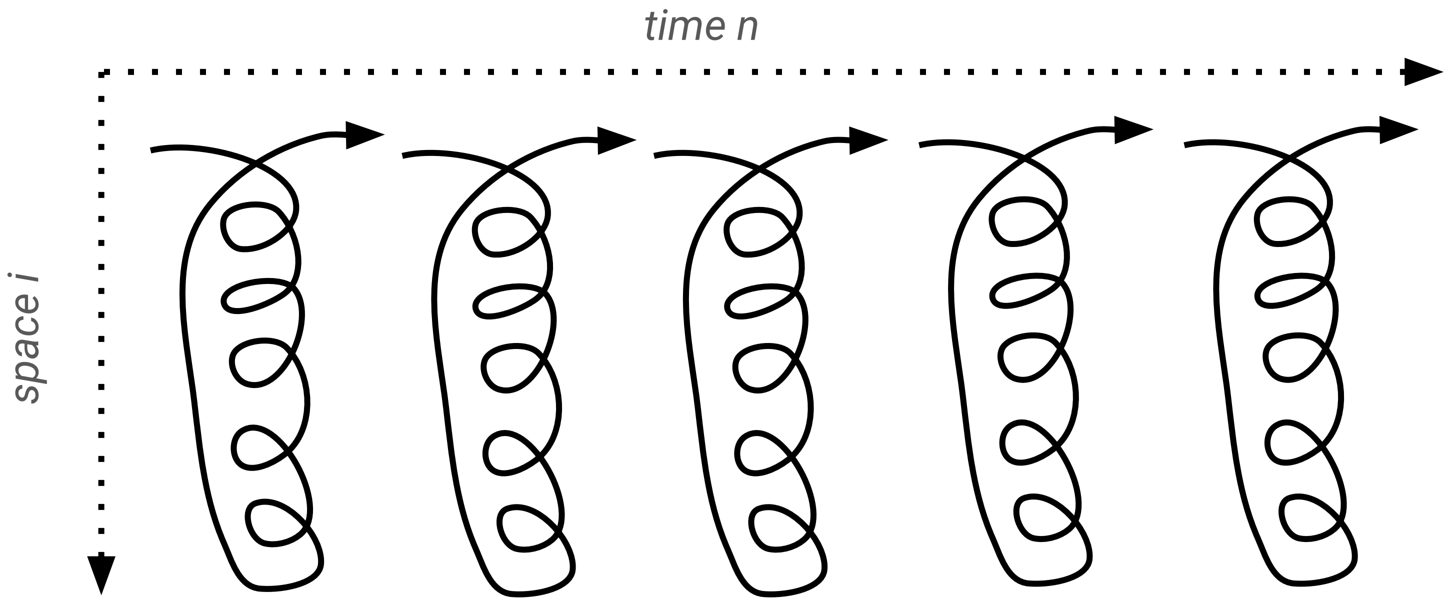
$$\frac{u\Delta t}{\Delta x} \leq 1$$

# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**

- the power of loops!

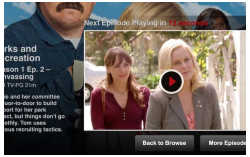
$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$



# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**
- the power of loops!

$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$

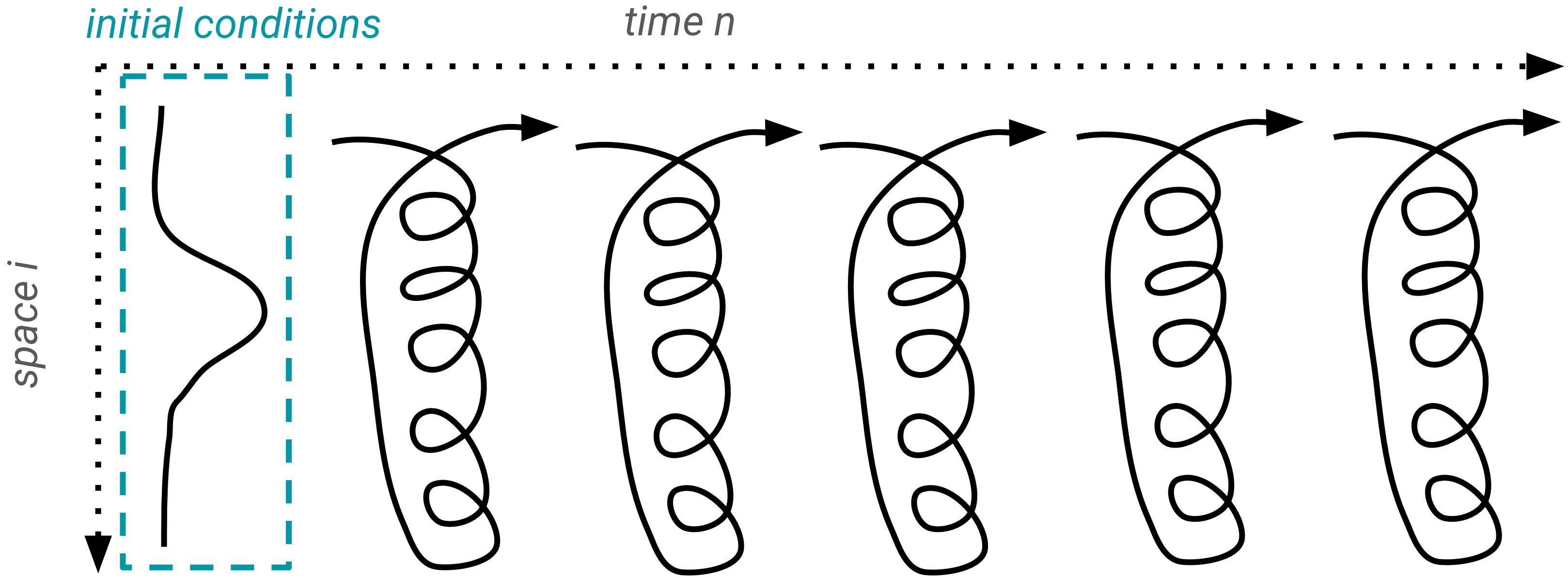



# Applying loops for water quality modeling

**FTCS (forward in time, centered in space)**

- the power of loops!

$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta z^2}$$



# Applying loops for water quality modeling

$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1} - 2C_i + C_{i-1}}{\Delta z^2}$$

*i* is space index, *n* is time index

```
> time = 100
> space = 100
> conc <- matrix(0, nrow = space, ncol = time)
# our results in a matrix: 100 seconds times 100 m over the depth
> K = 0.5 # diffusion coefficient, unit: m2/s
> dx = 1 # our spatial step, unit: m
> dt = 1 # our time step, unit: s
> conc[, 1] = dnorm(seq(1,100,1), mean = 50, sd = 0.1) * 100
# initial conc. is defined vertically through a normal distribution, unit: -

> for (n in 2:ncol(conc)){ # time index
      for (i in 2:(nrow(conc)-1)){ # space index
        conc[i, n] = conc[i, n-1] + K * dt / dx**2 * (conc[i+1, n-1] - 2 * conc[i, n-1] +
     conc[i-1, n-1]) # our FTCS schema
      }
  }
```

# Applying loops for water quality modeling

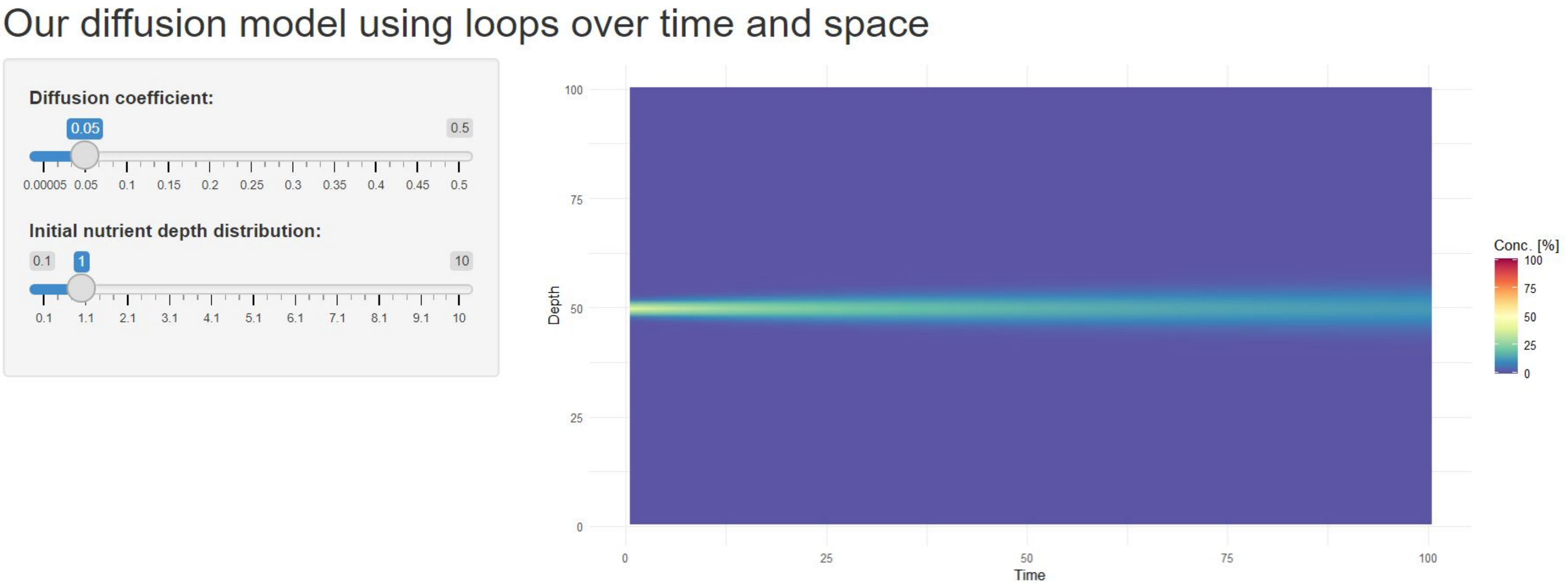
$$C_i^{n+1} = C_i^n + K\Delta t \frac{C_{i+1} - 2C_i + C_{i-1}}{\Delta z^2}$$

*i* is space index, *n* is time index

```
> time = 100
> space = 100
> conc <- matrix(0, nrow = space, ncol = time)
# our results in a matrix: 100 seconds times 100 m over the depth
> K = 0.5 # diffusion coefficient, unit: m2/s
> dx = 1 # our spatial step, unit: m
> dt = 1 # our time step, unit: s
> conc[, 1] = dnorm(seq(1,100,1), mean = 50, sd = 0.1) * 100
# initial conc. is defined vertically through a normal distribution, unit: -

> for (n in 2:ncol(conc)){ # time index
      for (i in 2:(nrow(conc)-1)){ # space index
        conc[i, n] = conc[i, n-1] + K * dt / dx**2 * (conc[i+1, n-1] - 2 * conc[i, n-1] +
     conc[i-1, n-1]) # our FTCS schema
      }
  }
```

# Applying loops for water quality modeling

- test our loop model: [shorturl.at/asx56](shorturl.at/asx56)
- play around with the diffusion coefficient and initial concentration distribution



# Downfall of loops

- loops are often not recommended for programming!
  (1) **memory inefficient** → reallocates output every time

```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector,
x * 2)}
```

# Downfall of loops

- loops are often not recommended for programming!
    (1) **memory inefficient** → reallocates output every time

    (2) **slow** → vectorized alternatives are faster

```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector,
x * 2)}
```

```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
    x <- x + item
}
```

or

```
> x = sum(seq(i,n))
```

# Downfall of loops

- loops are often not recommended for programming!
  (1) **memory inefficient** → reallocates output every time
  (2) **slow** → vectorized alternatives are faster
  (3) **endless loops** → can easily happen with `while`-loops

```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector,
x * 2)}
```

```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
    x <- x + item
}
```

or

```
> x = sum(seq(i,n))
```

# Downfall of loops

- loops are often not recommended for programming!
  (1) **memory inefficient** → reallocates output every time
  (2) **slow** → vectorized alternatives are faster
  (3) **endless loops** → can easily happen with `while`-loops
- **but**: if speed is not always the main issue, loops can be easier to understand + easy to implement ideas
- **and**: need loops for recursive relationships, e.g., when temporal processes depend on previous ones

```
> vector <- c(1, 2, 3)
> for (item in vector) {
      vector <- c(vector,
x * 2)}
```

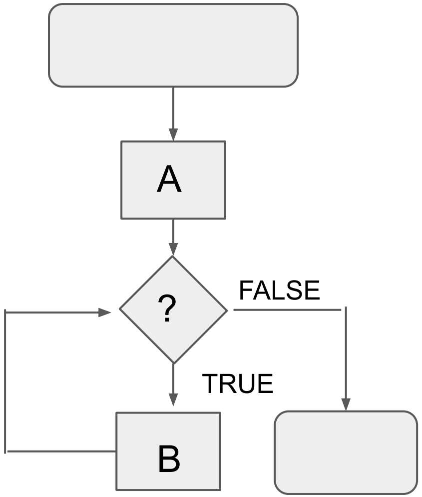
```
> i = 1
> n = 5
> x = 0
> for (item in i:n){
    x <- x + item
}
```

or

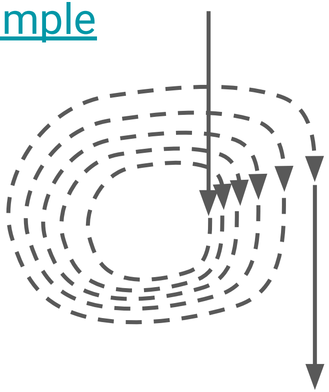
```
> x = sum(seq(i,n))
```

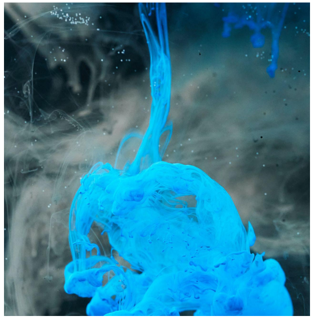
# Summing up

- **control flow** in essential for scientific programming
- loops in R are either **`for`** or **`while`**
- loops can be used to model a dynamic system
- potential shortcomings: **memory-inefficient** and **slow**
- useful to test ideas & organize workflows → loops are a ‘universal’ concept
- **material**: https://github.com/robertladwig/1DDiffusionExample



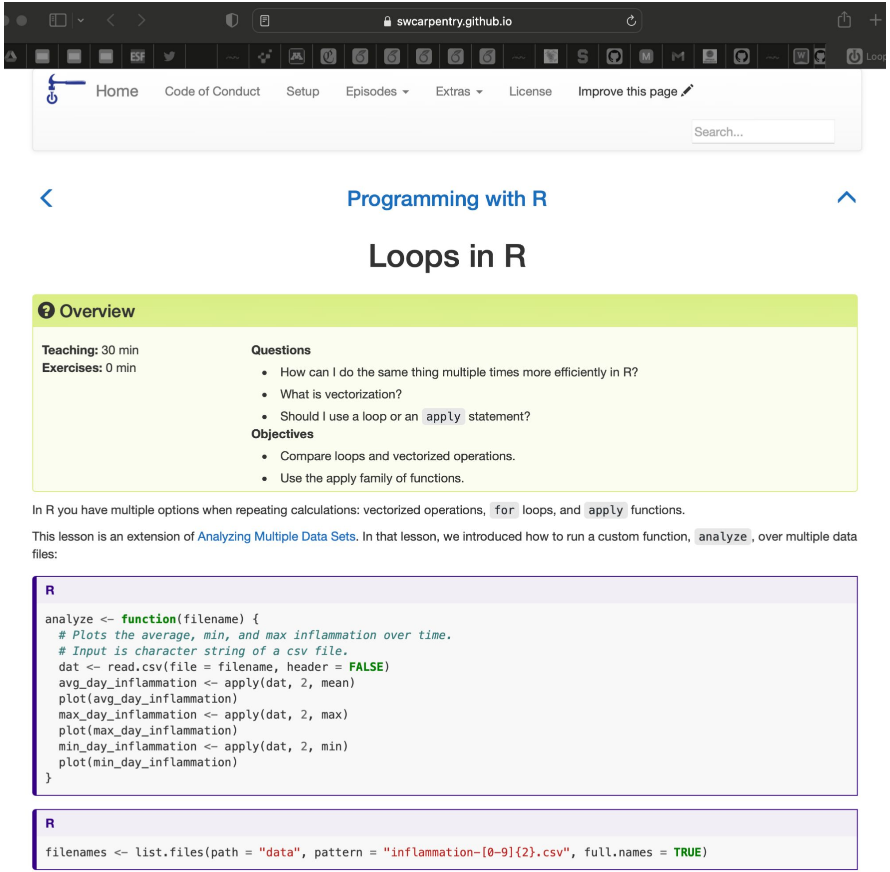

```
for (item in vector){

    perform_action

}
```





# Assignments

software carpentry: https://swcarpentry.github.io/r-novice-inflammation/15-supp-loops-in-depth/


swcarpentry.github.io

Home | Code of Conduct | Setup | Episodes | Extras | License | Improve this page

Search...

Programming with R

## Loops in R

**Overview**

**Teaching:** 30 min
**Exercises:** 0 min

**Questions**

- How can I do the same thing multiple times more efficiently in R?
- What is vectorization?
- Should I use a loop or an `apply` statement?

**Objectives**

- Compare loops and vectorized operations.
- Use the apply family of functions.

In R you have multiple options when repeating calculations: vectorized operations, `for` loops, and `apply` functions.

This lesson is an extension of Analyzing Multiple Data Sets. In that lesson, we introduced how to run a custom function, `analyze`, over multiple data files:

R
```
analyze <- function(filename) {
  # Plots the average, min, and max inflammation over time.
  # Input is character string of a csv file.
  dat <- read.csv(file = filename, header = FALSE)
  avg_day_inflammation <- apply(dat, 2, mean)
  plot(avg_day_inflammation)
  max_day_inflammation <- apply(dat, 2, max)
  plot(max_day_inflammation)
  min_day_inflammation <- apply(dat, 2, min)
  plot(min_day_inflammation)
}
```

R
```
filenames <- list.files(path = "data", pattern = "inflammation-[0-9]{2}.csv", full.names = TRUE)
```

# Applying loops for water quality modeling

- applying Taylor expansion: $$\frac{dC}{dt} = K\frac{d^2C}{dz^2}$$

- forwards:
- backwards:

$$C_{i+1} = C_i + \Delta z\frac{\partial C}{\partial z} + \frac{\Delta z^2}{2!}\frac{\partial^2 C}{\partial z^2} + \frac{\Delta z^3}{3!}\frac{\partial^3 C}{\partial z^3} + O(\Delta z^4)$$

$$C_{i-1} = C_i - \Delta z\frac{\partial C}{\partial z} + \frac{\Delta z^2}{2!}\frac{\partial^2 C}{\partial z^2} - \frac{\Delta z^3}{3!}\frac{\partial^3 C}{\partial z^3} + O(\Delta z^4)$$

- sum them up to get:
- and re-arrange:

$$C_{i+1} + C_{i-1} = 2C_i + \Delta z^2\frac{\partial^2 C}{\partial z^2} + O(\Delta z^4)$$

$$\frac{\partial^2 C}{\partial z^2} = \frac{C_{i+1} - 2C_i + C_{i-1}}{\Delta z^2} + O(\Delta z^4)$$

# Applying loops for water quality modeling

- applying Taylor expansion: $$\frac{dC}{dt} = K\frac{d^2C}{dz^2}$$

- *i* is space index, *n* is time index $$\frac{dC}{dt} = K\frac{C_{i+1} - 2C_i + C_{i-1}}{\Delta z^2}$$

- forward differencing in time: $$\frac{dC}{dt} = \frac{C_i^{n+1} - C_i^n}{\Delta t}$$

$$\boxed{C_i^{n+1} = C_i^n + K\Delta t\frac{C_{i+1} - 2C_i + C_{i-1}}{\Delta z^2}}$$

**FCTS (forward in time, central in space)**