

- [Home](#)
- [Exercícios](#)
- [Trabalho final](#)

A função em R encontra-se abaixo.

Notas:

- Para execução da função **é necessário conexão com a internet**;
- A função requer os pacotes “**jsonlite**” e “**maps**” instalados;
- O retorno de um grande número de observações provoca demora na execução da função.;
- Na proposta, eu havia indicado que o parâmetro '**captive**' seria por padrão FALSE. Aqui ele não tem valor padrão; os espécimes cativos e não cativos são retornados;
- Na proposta, eu havia indicado que o parâmetro '**geo**' seria por padrão FALSE. Aqui ele não tem valor padrão; as observações com ou sem dados de latitude e longitude são retornadas;
- Na proposta, eu havia indicado que o parâmetro '**quality_grade**' seria por padrão RESEARCH. Aqui ele não tem valor padrão; as observações podem ser dos três tipo existentes.

```
inaturalist = function(taxon_name = NULL,
  place = NULL,
  captive = NULL,
  geo = NULL,
  observed_on = NULL,
  quality_grade = NULL,
  date = NULL,
  circle = NULL,
  maxresults = 500) {
  # Apenas 'maxresults' tem valor default, os demais argumentos são definidos
  # inicialmente como nulos

  if (is.null(c(taxon_name, place, captive, geo, observed_on,
    quality_grade,
    date, circle))) {
    # Verifica se pelo menos um argumento foi informado, com exceção de
    # 'maxresults'

    stop ("Nenhum parâmetro foi informado!")
    # Mensagem de erro se nenhum argumento for informado
  }

  library(jsonlite)
  # Carrega biblioteca que permite acessar os dados da base de dados remota

  first = TRUE
  # Variável de controle para definir o primeiro parâmetro a ser inserido na
  # URL de busca. Apenas o primeiro não é antecedido por "&"
  url = "http://api.inaturalist.org/v1/observations?"
  # URL base para a busca das observações

  if (!is.null(taxon_name)) {
    # Verifica se o argumento 'taxon_name' foi informado
```

```
    if (class(taxon_name) != "character") {  
# Verifica se a classe do argumento 'taxon_name' e "character"  
    stop ("Valor do parâmetro 'taxon_name' inválido!")  
# Mensagem de erro se a classe do argumento 'taxon_name' nao for "character"  
    }  
  
    option = ifelse(first, "taxon_name=", "&taxon_name=")  
# Se for o primeiro parametro a ser inserido na URL, adiciona sem '&', do  
# contrario, adiciona com '&'  
    first = FALSE  
# Define que o primeiro parametro ja foi informado, assim, os proximos  
# parametros serao adicionados sem '&'  
  
    taxon_name = gsub(" ", "%20", tolower(taxon_name))  
# Troca espaços por %20, de acordo com as regras de criacao de URL's  
    url = paste(url, option, taxon_name, sep="")  
# Combina o argumento na URL de busca  
    }  
  
    if (!is.null(place)) {  
# Verifica se o argumento 'place' foi informado  
    if (class(place) != "character") {  
# Verifica se a classe do argumento 'place' e "character"  
    stop ("Valor do parâmetro 'place' inválido!")  
# Mensagem de erro se a classe do argumento 'place' nao for "character"  
    }  
  
    places = c(id=NA, name=NA)  
# Estrutura a variavel que guardara os locais que serao buscados  
  
    placex = gsub("[~`^]", "", iconv(place, to="ASCII//TRANSLIT"))  
# Regex que remove os acentos graficos do argumento ''  
    url_place =  
paste("http://api.inaturalist.org/v1/places/autocomplete?q=",  
    gsub(" ", "%20", tolower(placex)), sep="")  
# Prepara a URL de busca de locais (nao confundir com a URL de busca das  
# observacoes)  
  
    data_place = fromJSON(url_place)  
# Realiza a busca de locais apenas para pegar o total encontrados  
  
    url_place = paste(url_place, "&per_page=", data_place$total_results,  
sep="") # Redefine a URL de busca de locais para trazer todos de uma vez  
# (por padrao, o total e dividido por pagina e seria necessario percorrer cada  
# uma delas)  
    data_place = fromJSON(url_place)  
# Realiza novamente busca de locais, dessa vez retornando todos os itens  
# encontrados  
  
    if (data_place$total_results > 0) {
```

```
# Verifica se a busca encontrou retornou algo
  places = data_place$results[c("id", "name")]
# Armazena apenas os id's e os nomes dos locais
  places = places[tolower(gsub("[~'`^]", "", iconv(places$name,
    to="ASCII//TRANSLIT")) == tolower(placex),]
# Mantem apenas os locais em que os nomes casam com o argumento informado
( nao considera acentos)

  option = ifelse(first, "place_id=", "&place_id=")
# Verifica se 'local' e o primeiro parametro informado
  first = FALSE
# Define que o primeiro parametro ja foi informado

  places_id = paste(places$id, collapse=",")
# Separa os id's dos locais encontrados, separados por virgula
  url = paste(url, option, places_id, sep="")
# Combina os id's na URL de busca de observacoes
  } else {
    warning(paste("Nenhum local encontrado para ", place,
      ". A busca não filtrará as observações por local!", sep=""))
# Notifica que nao foi encontrado um local que case com o argumento
informado e que a busca de observacoes sera realizada sem o parametro
  }
}

if (!is.null(captive)) {
# Verifica se o argumento 'captive' foi informado
  if (class(captive) != "logical") {
# Verifica se a classe do argumento 'captive' e "logical"
    stop ("Valor do parâmetro 'captive' inválido!")
# Mensagem de erro se a classe do argumento nao for "logical"
  }

  option = ifelse(first, "captive=", "&captive=")
# Se for o primeiro parametro a ser inserido na URL, adiciona sem &
  first = FALSE
# Define que o primeiro parametro ja foi informado

  url = paste(url, option, tolower(captive), sep="")
# Combina o argumento na URL de busca, com letras minusculas
  }

if (!is.null(geo)) {
# Verifica se o argumento 'geo' foi informado
  if (class(geo) != "logical") {
# Verifica se a classe do argumento 'geo' e "logical"
    stop ("Valor do parâmetro 'geo' inválido!")
# Mensagem de erro se a classe do argumento 'geo' nao for "logical"
  }

  option = ifelse(first, "geo=", "&geo=")
```

```
# Se for o primeiro parametro a ser inserido na URL, adiciona sem &
    first = FALSE
# Define que o primeiro parametro ja foi informado

    url = paste(url, option, tolower(geo), sep="")
# Combina o argumento na URL de busca, com letras minusculas
}

    if (!is.null(observed_on)) {
# Verifica se o argumento foi informado
        if (class(observed_on) != "Date") {
# Verifica se a classe do argumento 'observed_on' e "Date"
            stop ("Valor do parâmetro 'observed_on' inválido!")
# Mensagem de erro se a classe do argumento nao for "Date"
        }

        option = ifelse(first, "observed_on=", "&observed_on=")
# Se for o primeiro parametro a ser inserido na URL, adiciona sem &
        first = FALSE
# Define que o primeiro parametro ja foi informado

        url = paste(url, option, format(observed_on, "%Y-%m-%d"), sep="")
# Combina o argumento na URL de busca, no formato %Y-%m-%d
    }

    if (!is.null(quality_grade)) {
# Verifica se o argumento 'quality_grade' foi informado
        if ((class(quality_grade) != "character") | !(tolower(quality_grade)
            %in% c("casual", "needs_id", "research"))) {
# Verifica se a classe do argumento e se ele esta dentro das opcoes
# disponiveis

            stop ("Valor do parâmetro 'quality_grade' inválido!")
# Mensagem de erro se algum problema for encontrado com o argumento
        }

        option = ifelse(first, "quality_grade=", "&quality_grade=")
# Se 'quality_grade' for o primeiro parametro a ser inserido na URL,
# adiciona sem &
        first = FALSE
# Define que o primeiro parametro ja foi informado

        url = paste(url, option, tolower(quality_grade), sep="")
# Combina o argumento na URL de busca
    }

    if (!is.null(date)) {
# Verifica se o argumento 'date' foi informado
        if (class(date) != "list" | length(date) > 3
            | class(unlist(date)) != "numeric") {
```

```
# Verifica se o argumento e do tipo lista, se tem no maximo 3 elementos e se
o tipo de dados deles e "numeric"

    stop ("Valor do parâmetro 'date' inválido!")
# Mensagem de erro se algum problema for encontrado com o argumento
}

    if (!is.null(date$day)) {
# Verifica se a opcao 'day' foi informada
        if (!date$day %in% 1:31) {
# Verifica se a opcao 'day' esta entre 1 e 31
            stop ("Valor do parâmetro 'day' em 'date' inválido!")
# Mensagem de erro se a opcao 'day' estiver em intervalo invalido
        }

        option = ifelse(first, "day=", "&day=")
# Verifica se a opcao 'day' e o primeiro parametro de busca
        url = paste(url, option, date$day, sep="")
# Combina a opcao 'day' na URL de busca
        first = FALSE
# Define que o primeiro parametro ja foi informado
    }

    if (!is.null(date$month)) {
# Verifica se a opcao 'month' foi informada
        if (!date$month %in% 1:12) {
# Verifica se a opcao 'month' esta entre 1 e 12
            stop ("Valor do parâmetro 'month' em 'date' inválido!")
# Mensagem de erro se a opcao 'month' estiver em intervalo invalido
        }

        option = ifelse(first, "month=", "&month=")
# Verifica se a opcao 'month' e o primeiro parametro de busca
        url = paste(url, option, date$month, sep="")
# Combina a opcao 'month' na URL de busca
        first = FALSE
# Define que o primeiro parametro ja foi informado
    }

    if (!is.null(date$year)) {
# Verifica se a opcao 'year' foi informada
        option = ifelse(first, "year=", "&year=")
# Verifica se a opcao 'year' e o primeiro parametro de busca
        url = paste(url, option, date$year, sep="")
# Combina a opcao 'year' na URL de busca
        first = FALSE
# Define que o primeiro parametro ja foi informado
    }
}

if (!is.null(circle)) {
```

```
# Verifica se o argumento 'circle' foi informado
    if ((class(circle) != "list") | (length(circle) != 3
        | class(unlist(circle)) != "numeric")) {
# Verifica se o argumento e do tipo lista, se tem 3 elementos e se o tipo de
dados deles e "numeric"

        stop ("Valor do parâmetro 'circle' inválido!")
# Mensagem de erro se algum problema for encontrado com o argumento
    }

    if (!is.null(circle$lat)) {
# Verifica se a opcao 'lat' foi informada
        option = ifelse(first, "lat=", "&lat=")
# Verifica se a opcao 'lat' e o primeiro parametro de busca
        url = paste(url, option, circle$lat, sep="")
# Combina a opcao 'lat' na URL de busca
        first = FALSE
# Define que o primeiro parametro ja foi informado. Nesse caso, como as tres
opcoes sao obrigatorias, se este foi o primeiro, nao sera necessario
verificar nos proximos
    } else {
        stop ("Valor do parâmetro 'circle' inválido!")
# Se nao tem a opcao 'lat', ha algo de errado no parametro, pois nesse ponto
sabemos que existem 3 objetos no argumento
    }

    if (!is.null(circle$lng)) {
# Verifica se a opcao 'lng' foi informada
        url = paste(url, "&lng=", circle$lng, sep="")
# Combina a opcao 'lng' na URL de busca
    } else {
        stop ("Valor do parâmetro 'circle' inválido!")
# Se nao tem a opcao 'lng', ha algo de errado no parametro, pois nesse ponto
sabemos que existem 3 objetos no argumento
    }

    if (!is.null(circle$radius)) {
# Verifica se a opcao 'radius' foi informada
        url = paste(url, "&radius=", circle$radius, sep="")
# Combina a opcao 'radius' na URL de busca
    } else {
        stop ("Valor do parâmetro 'circle' inválido!")
# Se nao tem a opcao 'radius', ha algo de errado no parametro, pois nesse
ponto sabemos que existem 3 objetos no argumento
    }
}

if (class(maxresults) != "numeric") {
# Verifica se o argumento foi informado
    stop ("Valor do parâmetro 'maxresults' inválido!")
}
```

```

# Mensagem de erro se o tipo do argumento nao for "numeric"
}

url = paste(url, "&order=desc&order_by=created_at", sep="")
# Define a ordenacao dos dados que serao encontrados: decrescente por data
de criacao

data = fromJSON(url)
# Realiza a busca das observacoes na base de dados

if (data$total_results == 0) {
# Verifica se a busca retornou observacoes
  stop ("Não encontramos observações com os filtros passados!")
# Exibe mensagem de erro se nao houver observacoes
}

data_to_return = c(id=NA, observed_on=NA, captive=NA,
  quality_grade=NA, latitude=NA, longitude=NA, taxon_name=NA,
place_guess=NA,
  url=NA)
# Prepara formato da tabela que sera retornada (essa linha sera removida
posteriormente)

i = 1;
# Inicia contador do loop
while(TRUE) {
# Loop infinito (o criterio de parada e feito dentro da repeticao)
  if (i > ceiling(data$total_results / data$per_page)) {
# Os dados sao divididos em paginas. Aqui e verificado se todas elas ja
foram acessadas. Esse e o criterio de parada
    break
# Para a repeticao se todas as paginas de resultado ja foram acessadas
  }

  url_page = paste(url, "&page=", i, sep="")
# Prepara a URL para acessar cada pagina individualmente. O valor de 'i'
indica a pagina atual

  data = fromJSON(url_page)
# Refaz a busca para uma unica pagina

  dt1 = NULL
# Data.frame temporario para armazenar os dados da pagina
  dt1 = data$results[c("id", "observed_on", "captive",
"quality_grade")] # Seleciona algumas colunas da tabela de
resultado

  if (!is.null(nrow(data$results$geojson))) {
# Verifica se existe pelo menos um dado de latitude e longitude na pagina
data$results$geojson$coordinates[sapply(data$results$geojson$coordinates,
  is.null)] = NA

```

```
# Insere NA nos locais que nao ha dados de latitude e longitude

coordinates = unlist(data$results$geojson$coordinates)
# Transforma a lista de coordenadas em vetor

if (sum(is.na(coordinates)) > 0) {
# Verifica se existem NA's nas coordenadas
  indexes = which(is.na(coordinates))
# Pega os indices das posicoes com NA's
  for (j in rev(indexes)) {
# Percorre os NA's na ordem inversa
    coordinates = append(coordinates, NA, j)
# Adiciona mais um NA logo em seguida. Tinhamos apenas um NA para a dupla
# latitude e longitude. Agora temos um valor NA para cada
  }
}

dt1$longitude = as.numeric(coordinates[seq(from=1, to=(2 *
nrow(dt1)),
  by=2)])
# Separa os indices impares das coordenadas para a coluna longitude

dt1$latitude = as.numeric(coordinates[seq(from=2, to=(2 *
nrow(dt1)),
  by=2)])
# Separa os indices pares das coordenadas para a coluna latitude
}

dt1$taxon_name = unlist(data$results$taxon$name)
# Transforma a lista de nomes dos taxos em vetor e adiciona no data.frame
temporario
dt1$place_guess = unlist(data$results$place_guess)
# Transforma a lista de locais em vetor e adiciona no data.frame temporario
dt1$url = unlist(data$results$uri)
# Transforma a lista de URL's das observacoes em vetor e adiciona no
data.frame temporario

data_to_return = rbind(data_to_return, dt1)
# Combina o data.frame temporario com os dados que serao retornados

if (i * data$per_page >= maxresults + 1) {
# Verifica se ja atingimos o valor maximo de observacoes definido em
'maxresults'
  data_to_return = data_to_return[1:(maxresults + 1),]
# Caso sim, remove os excedentes
  break
}

i = i + 1
# Incrementa o contador (indica a pagina)
```



```

    }

    data_to_return = data_to_return[-1,]
# Remove a primeira linha do datatable (criada no momento de preparacao com
NA's)
    rownames(data_to_return) = NULL
# Recalcula o numero das linhas depois da remocao da primeira linha

    pontos = data.frame(data_to_return$longitude, data_to_return$latitude)
# Separa os pontos para plotagem no mapa

    if (sum(complete.cases(pontos)) > 0) {
# Se existir observacoes georreferenciadas
        library(maps)
# Carrega a biblioteca para plotagem do mapa

        map("world", col="gray35")
# Define o mapa a ser exibido
        map.axes(bty="o", col="gray35", xaxt="n", yaxt="n")
# Define as bordas do plot, a cor da linha do mapa e remove os indices
        abline(h = 0, lty = 2, col="gray66")
# Adiciona a linha do equador
        points(pontos, col='red', pch=4, cex=0.3)
# Insere os pontos no mapa
    } else {
        warning ("Não encontramos observações com dados georreferenciados!")
# Mensagem de aviso se os dados retornados nao possuirem pontos
georreferenciados. Nesse caso o mapa nao e construido, so o data.frame e
retornado
    }

    return (data_to_return)
# Retorna as observacoes encontradas em um data.frame
}

```

Abaixo, o help da função.

inaturalist

package:unknown

R Documentation

Description:

A função realiza a busca de observações de espécimes registrados na base de dados do portal iNaturalist (<http://www.inaturalist.org>) através da sua API pública (<http://api.inaturalist.org>). A busca é realizada de acordo com os parâmetros passados para a função. Nenhum parâmetro é obrigatório, porém é necessário informar pelo menos um (com exceção de 'maxresults'). As observações encontradas são retornada em um **data.frame** e plotadas em um mapa. A função requer

os pacotes `"jsonlite"` e `"maps"` instalados.

Usage:

```
inaturalist(taxon_name, place, captive, geo, observed_on, quality_grade,  
date, circle, maxresults = 500)
```

Arguments:

`taxon_name`: um objeto do tipo `"character"` que informa o táxon dos espécimes que se deseja buscar;

`place`: um objeto do tipo `"character"` que informa a localização a que