WDR – Wykład nr 1

Wydział Informatycznych Technik Zarządzania WIT
studia I stopnia stacjonarne – semestr letni 2023/2024

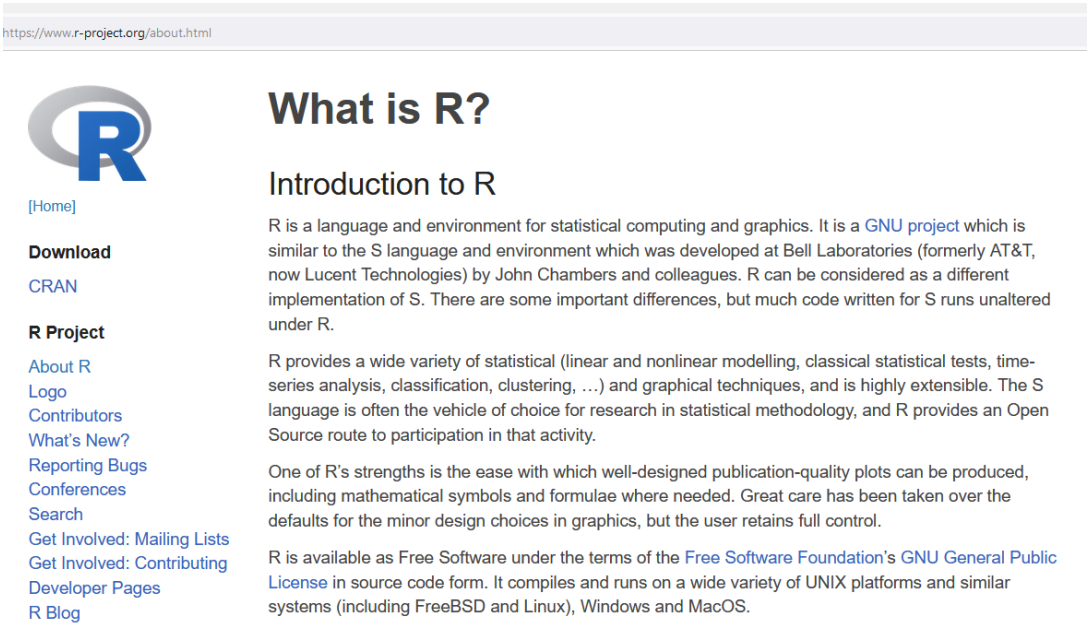
Środowisko R i program RStudio

Wprowadzenie

1. Korporacje, rządy, instytuty naukowe, użytkownicy Internetu generują informacje różnego rodzaju: dotyczące komunikacji międzyludzkiej, kultury, ekonomii, handlu, przemysłu, transportu, zdrowia, środowiska itd. Postęp technologiczny jaki dokonał się w naukach informacyjnych na przestrzeni ostatnich dekad, skutkuje tym, że dzisiejszy świat wytwarza więcej danych, niż jest ich w stanie efektywnie przetworzyć. Nie jest to spowodowane brakiem dostępności odpowiednich algorytmów i narzędzi, ale raczej trudnością w znalezieniu dobrze przygotowanych profesjonalistów.
2. Przekształcanie surowych danych (ilościowych, jakościowych, tekstu, itp.) na przyswajalną wiedzę jest celem działania m.in. analityków danych, specjalistów *business intelligence*, badaczy opinii i rynku, czy *data scientists*. Szczególnie istotna jest biegła umiejętność obsługi narzędzi informatycznych, które służą do przechowywania i przetwarzania informacji, wydobywania z nich wiedzy i prezentacji uzyskiwanych wyników – i to na poziomie, który nie tylko implikuje sprawne wykorzystanie istniejących rozwiązań, ile przede wszystkim umożliwia analizę, projektowanie, implementację, testowanie i wdrażanie własnych pomysłów.[1]

Czym jest R?

R jest środowiskiem oraz językiem programowania wykorzystywanym najczęściej do przeprowadzania obliczeń statystycznych i numerycznych, analizy danych, tworzenia raportów i wysokiej jakości grafiki.



Środowisko R powstało w 1997 roku na Uniwersytecie w Auckland w nowej Zelandii, jest podobne do środowiska i języka S zaprojektowanego w Bell Laboratories przez Johna Chambersa i jego kolegów. Pierwsza wersja R, która nie była już wersją testową, to wersja 1.0 wydana w 2000 roku, aktualnie dostępna jest wersja 4.3.2. Jądro R składa się z napisanej w językach C i Fortran implementacji znanego od 1976 roku i wciąż rozwijanego języka S.

R jest **wolnym oprogramowaniem** (ang. Free Software). **Licencja GNU GPL** (ang. General Public License) zezwala na wykorzystanie środowiska R również w zastosowaniach komercyjnych. Istnieje także jego komercyjna wersja, zoptymalizowana pod kątem wysoko wydajnych obliczeń na dużych zbiorach danych – Microsoft R.

[1] M. Gągolewski, *Programowanie w języku R. Analiza danych, obliczenia, symulacje*. PWN, Warszawa, 2016

Cechy języka i środowiska R

- Jest językiem *ogólnego zastosowania* – można w nim zaimplementować praktycznie każdy algorytm.
- Jest przeznaczony do pisania raczej *„małych" programów*, tj. takich, gdzie sednem jest wykonanie obliczeń bez zwracania uwagi na sposoby interakcji z użytkownikiem (którym zwykle jest ściśle określony specjalista).
- R cechuje się bardzo *zwięzłą składnią*, tzn. mało kodu daje duży rezultat.
- Jest językiem ***interpretowanym***, pozwala to na pracę w nim w sposób interaktywny, otrzymując prawie natychmiast wynik wykonywanych poleceń.
- Ma rozbudowane możliwości *generowania wysokiej jakości grafiki* (wykresy, diagramy) do wszelkiego rodzaju publikacji. Próbka możliwości na https://www.r-graph-gallery.com
- W repozytorium CRAN (Comprehensive R Archive Network) udostępnionych jest ok. 20 tysięcy **pakietów** rozszerzających możliwości bazowego R.
- R ma obszerną, łatwo dostępną *dokumentację*.

R jest często nazywany pakietem statystycznym. Jest tak z uwagi na olbrzymią liczbę dostępnych funkcji statystycznych. Możliwości R są jednak znacznie większe. Pod względem zastosowań podobne do niego są środowisko Matlab oraz język Python.

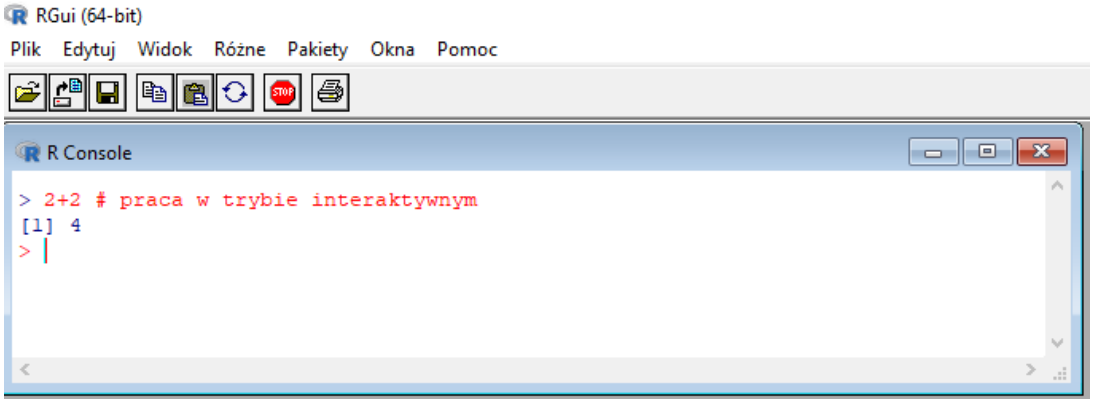
Instalacja R

Program działa na różnych systemach operacyjnych: Windows, Linux, Solaris i OS X.
Instalacja R: https://cloud.r-project.org/

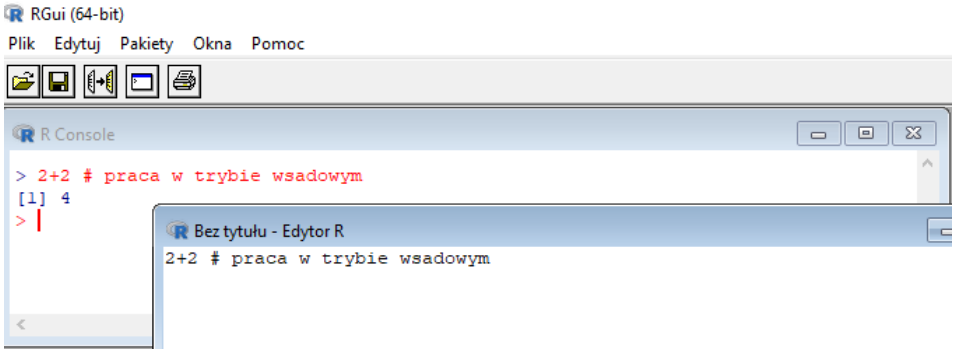
Organizacja pracy w R

R może pracować w dwóch trybach:

- w **trybie interaktywnym**, gdzie po każdym wydanym poleceniu otrzymujemy rezultat jego wykonania,



- w **trybie wsadowym**, w którym zlecamy środowisku R uruchomienie danego pliku źródłowego (skryptu), czyli pliku tekstowego najczęściej o rozszerzeniu .R, zawierającego kolejne polecenia języka R przeznaczone do wykonania.



W praktyce najczęściej korzysta się przemiennie z obu tych trybów.

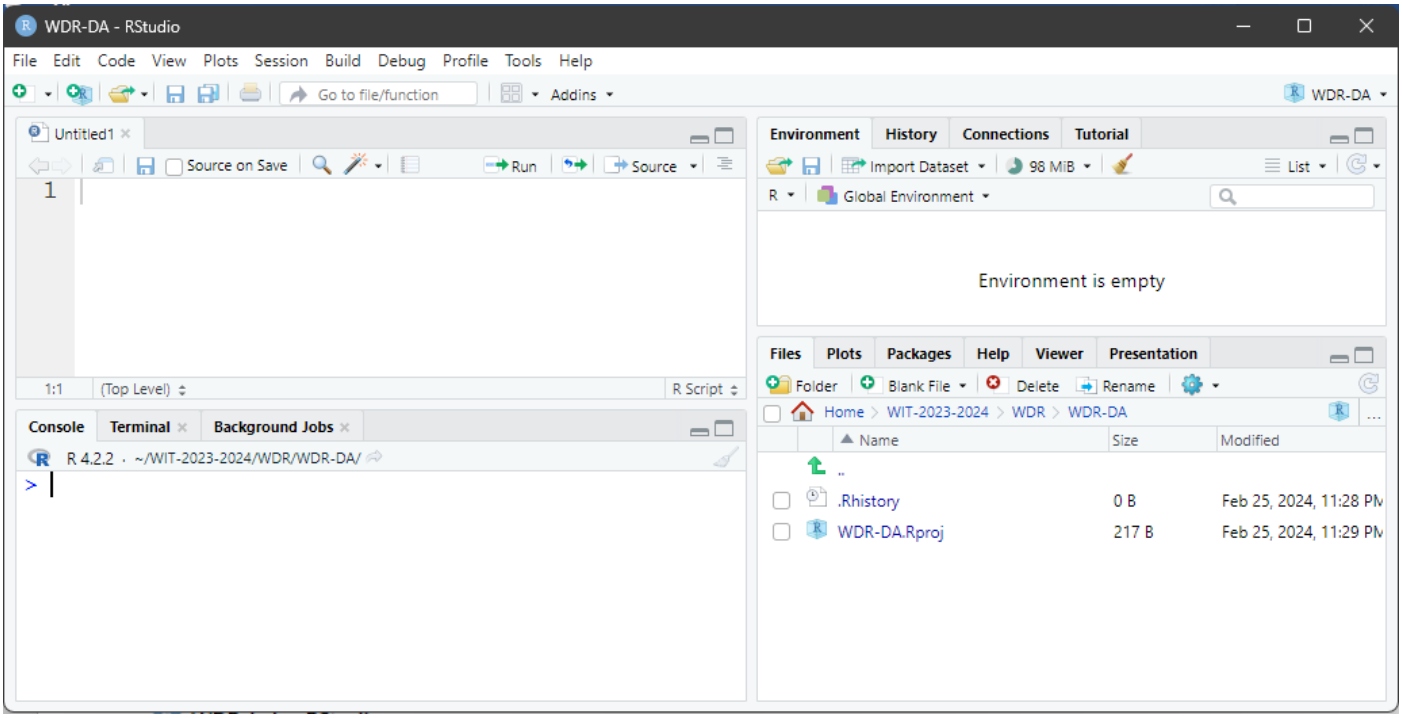
Program RStudio

RStudio jest *zintegrowanym środowiskiem programistycznym* (ang. IDE, *integrated development environment*) dla istniejącej już w systemie instalacji R.

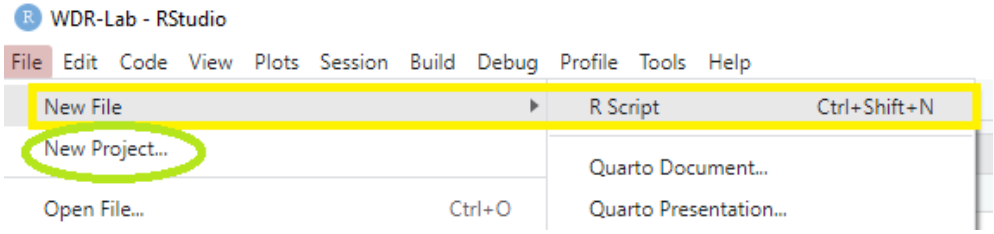
RStudio znakomicie ułatwia pracę ze środowiskiem R poprzez m.in.:

- liczne rozszerzenia możliwości konsoli,
- wygodne zarządzanie skryptami,
- zintegrowany system pomocy,
- narzędzia wspomagające zarządzanie generowanymi plikami graficznymi.

Otwarty, bezpłatny program **RStudio Desktop** można pobrać ze strony https://posit.co/downloads/



Lewe okno to konsola, pisząc w niej pracujemy w trybie interaktywnym. Chcąc pracować w trybie wsadowym tworzymy nowy plik tzw. **R Script**. Wygodnie jest wszystkie skrypty z zajęć laboratoryjnych przechowywać w tzw. projekcie np. z nazwą WDR-Lab ☺



New Project >> New Directory >> New Project >> Directory name: WDR-Lab (i ew. zmiana ścieżki do pliku z projektem) >> Create Project

KLASYFIKACJA TYPÓW OBIEKTÓW W JĘZYKU R

Typy podstawowe	Typy atomowe	wektor wartości logicznych	typ logical
		wektor wartości całkowitych	typ integer
		wektor wartości rzeczywistych	typ double
		wektor napisów	typ character
		wektor wartości zespolonych	typ complex
		wektor bajtów	typ raw
		typ pusty	typ NULL
	Typy o strukturze rekurencyjnej	lista (wektor uogólniony)	typ list
		funkcja	typ function
		środowisko	typ environment
	Typy reprezentujące nieobliczone wyrażenia języka R	nazwa (symbol)	typ name
		wywołanie	typ call
		ciąg wyrażeń	typ expression
Typy złożone		macierz i tablica	klasa matrix,array
		szereg czasowy	klasa ts
		czynnik	klasa factor
		ramka danych	klasa data frame
		formuła	klasa formula

Typy atomowe

1. Wektory wartości logicznych

W R zdefiniowane są dwie stałe logiczne: TRUE i FALSE[2].

Tworzenie wektorów przez:

- złączenie – funkcja c()
- replikację – funkcja rep()

```
> TRUE
[1] TRUE
> c(TRUE) # to samo co wyżej
[1] TRUE
> c(TRUE, FALSE, FALSE, TRUE)
[1]  TRUE FALSE FALSE  TRUE
> c(c(TRUE, FALSE), c(FALSE, TRUE))
[1]  TRUE FALSE FALSE  TRUE
>
```

```
> rep(TRUE, 3)
[1] TRUE TRUE TRUE
> ?rep
> rep(c(TRUE,FALSE),2)
[1]  TRUE FALSE  TRUE FALSE
> rep(c(TRUE,FALSE), times=2)
[1]  TRUE FALSE  TRUE FALSE
> rep(c(TRUE,FALSE), each=2)
[1]  TRUE  TRUE FALSE FALSE
> rep(c(TRUE,FALSE), length.out=3)
[1]  TRUE FALSE  TRUE
> rep(c(TRUE,FALSE), , 3)
[1]  TRUE FALSE  TRUE
> rep(c(TRUE,FALSE), times=2, each=2)
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
>
```

Usage

```
rep(x, ...)

rep.int(x, times)

rep_len(x, length.out)
```

Arguments

x	a vector (of any mode including a list) or a factor or (for rep only) a POSIXct or POSIXlt or Date object; or an S4 object containing such an object.
...	further arguments to be passed to or from other methods. For the internal default method these can include:
times	an integer-valued vector giving the (non-negative) number of times to repeat each element if of length length(x), or to repeat the whole vector if of length 1. Negative or NA values are an error. A double vector is accepted, other inputs being coerced to an integer or double vector.
length.out	non-negative integer. The desired length of the output vector. Other inputs will be coerced to a double vector and the first element taken. Ignored if NA or invalid.
each	non-negative integer. Each element of x is repeated each times. Other inputs will be coerced to an integer or double vector and the first element taken. Treated as 1 if NA or invalid.

Funkcja rep() ma 4 parametry o nazwach: x, times, length.out, each. Wartości tych parametrów podajemy korzystając z ich pozycji na liście parametrów funkcji lub przez odwołanie się do nazwy parametru (lub części nazwy, o ile jednoznacznie wskazuje na dany parametr).

[2] Dla wygody zostały także określone ich synonimy, odpowiednio, T i F. W odróżnieniu od TRUE i FALSE nie są one słowami kluczowymi, więc łatwo je nadpisać. Z tego względu nie zaleca się ich stosować jako stałe logiczne.

2. Wektory liczbowe

Każdy ciąg cyfr uznawany jest za stałą liczbową. Części dziesiętne oddziela się kropką.

```
> 1  # 1-elementowy wektor liczbowy
[1] 1
> c(1, -2, +3, 4., -.5, 6)
[1]  1.0 -2.0  3.0  4.0 -0.5  6.0
> rep(3.2, 3)
[1] 3.2 3.2 3.2
> 1.2e-2 # notacja naukowa
[1] 0.012
> 1.2e-16
[1] 1.2e-16
```

Tworzenie ciągów arytmetycznych przez:

- operator : (dwukropek)
- funkcję seq()

```
> -2:2
[1] -2 -1  0  1  2
> 5:1
[1] 5 4 3 2 1
> c(-2:2, 5:1)
 [1] -2 -1  0  1  2  5  4  3  2  1
> seq(1,10,1)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(10,1,-2)
[1] 10  8  6  4  2
> seq(0, 1, length.out=5)
[1] 0.00 0.25 0.50 0.75 1.00
```

3. Wektory napisów

Do wprowadzenia napisów (ciągów znaków) służą cudzysłów lub apostrofy.

```
> "Idzie wiosna"
[1] "Idzie wiosna"
> c("Idzie wiosna", 'Swieci slonce')
[1] "Idzie wiosna"  "Swieci slonce"
```

Długość wektora oblicza funkcja length().

```
> length(c("Idzie wiosna", 'Swieci slonce'))
[1] 2
```

Definiując napis, niektórych znaków nie da się wprowadzić wprost – dotyczy to na przykład cudzysłowu, czy apostrofu. W związku z tym przyjęto, że pewne znaki poprzedzone odwróconym ukośnikiem (ang. backslash) będą miały specjalne znaczenie.

```
> "Napisy twórz "tak" albo 'tak'"
BŁĄD: nieoczekiwany symbol w ""Napisy twórz "tak"
> "Napisy twórz \"tak\" albo 'tak'"
[1] "Napisy twórz \"tak\" albo 'tak'"
> "Idzie \n wiosna"
[1] "Idzie \n wiosna"
> cat("Napisy twórz \"tak\" albo 'tak'")
Napisy twórz "tak" albo 'tak'
> print("Napisy twórz \"tak\" albo 'tak'")
[1] "Napisy twórz \"tak\" albo 'tak'"
> cat("Napisy twórz \"tak\" \n albo 'tak'")
Napisy twórz "tak"
 albo 'tak'
>
```

Wybrane znaki specjalne					
\n	nowy wiersz	\t	tabulator	\\	odwrócony ukośnik
\r	powrót karetki do początku wiersza	\b	usunięcie znaku poprzedzającego	\' lub \"	apostrof, czudzysłów

Typ podstawowy każdego obiektu R możemy poznać wywołując funkcję typeof().

```
> typeof(TRUE)
[1] "logical"
> typeof(c(1,2,3))
[1] "double"
> typeof("Idzie wiosna")
[1] "character"
```

Podobną funkcją jest mode(). Wynik przez nią zwracany często pokrywa się z z tym z typeof().

```
> mode(TRUE)
[1] "logical"
> mode(c(1,2,3))
[1] "numeric"
> mode("Idzie wiosna")
[1] "character"
```

4. **Wektory wartości całkowitych**

Wektor liczbowy w pamięci komputera może być reprezentowany przy użyciu dwóch natępujących typów podstawowych:

- całkowite (integer),
- rzeczywiste (double).

Mimo to w R wprowadzane stałe liczbowe i wektory tworzone przez wywołanie funkcji c(), rep(), seq() są traktowane najczęściej jako zmienne typu rzeczywistego (nawet gdy część ułamkowa jest równa zero).

```
> c(typeof(1), mode(1))
[1] "double"  "numeric"
> c(typeof(1.5), mode(1.5))
[1] "double"  "numeric"
> c(typeof(1L), mode(1L))
[1] "integer" "numeric"
> c(typeof(1:5), mode(1:5))
[1] "integer" "numeric"
```

5. **Wektory bajtów***

Można w nich przechowywać wyłącznie wartości ze zbioru {0,1, …, 255}. Tego rodzaju obiekty mogą się przydać np. w przypadku przetwarzania napisów w pewnych kodowaniach bądź plików binarnych. Elementy wektora bajtów są wypisywane na konsoli w notacji szesnastkowej.

6. **Wektory wartości zespolonych***

```
> 1+2i
[1] 1+2i
> c(typeof(1+2i), mode(1+2i))
[1] "complex" "complex"
> sqrt(-4)
[1] NaN
Komunikat ostrzegawczy:
W poleceniu 'sqrt(-4)': wyprodukowano wartości NaN
> sqrt(as.complex(-4))
[1] 0+2i
```

7. **Hierarchia i uzgadnianie typów**

Wektory atomowe przechowują elementy jednego ściśle określonego typu: mają jednorodną, atomową strukturę. Co się stanie, jeśli spróbujemy stworzyć wektor przy użyciu funkcji c() do złączenia wartości z różnych dziedzin?

```
> c(FALSE, 1L, 2.4, 2i, "siedem")
[1] "FALSE"  "1"      "2.4"    "0+2i"   "siedem"
> c(FALSE, 1L, 2.4, 2i)
[1] 0.0+0i 1.0+0i 2.4+0i 0.0+2i
> c(FALSE, 1L, 2.4)
[1] 0.0 1.0 2.4
> c(FALSE, 1L)
[1] 0 1
> c(FALSE)
[1] FALSE
```

Gdy łączymy ze sobą wektory różnych rodzajów, **R uzgodni ich typ** do najbardziej ogólnego typu (uzgadnianie, koercja). Z powyższego przykładu widać jaka jest hierarchia typów R od najbardziej do najmniej szczegółowego:

1) wektor wartości logicznych (typ logical),
2) wektor wartości całkowitych (typ integer),
3) wektor wartości rzeczywistych (typ double),
4) wektor wartości zespolonych (typ complex),
5) wektor napisów (typ character).

Rzutowanie typów

Wszystkie wektory można także w sposób jawny zrzutować na wektory innych typów przy użyciu funkcji:

- as.character(),
- as.complex(),
- as.double(),
- as.numeric(),
- as.integer(),
- as.raw(),
- as.logical().

```
> as.numeric(c(TRUE, FALSE))
[1] 1 0
> as.character(c(TRUE, FALSE))
[1] "TRUE"  "FALSE"
> as.logical(-2:2)
[1]  TRUE  TRUE FALSE  TRUE  TRUE
> as.logical(c("FALSE", "F", "f", "T"))
[1] FALSE FALSE    NA  TRUE
```

Rzutowanie liczby rzeczywistej do całkowitej następuje przez obcięcie jej części ułamkowej.

```
> as.integer(c(1.4, 1.99, -2.99, 0.777))
[1]  1  1 -2  0
> as.integer(c("1.4", "1.99", "-2.99?", "1e5"))
[1]      1      1     NA 100000
```

Do rzutowania typów można zastosować także ogólniejszą funkcję as.vector()

```
> as.vector(c(TRUE, FALSE), "numeric")
[1] 1 0
```

Sprawdzanie typu obiektu

Istnieją funkcje do sprawdzenia, czy dany obiekt jest określonego typu:

- is.character(),
- is.complex(),
- is.double(),
- is.numeric(),
- is.integer(),
- is.raw(),
- is.logical()

oraz is.vector(), is.atomic().

```
> is.integer(1:5)
[1] TRUE
> is.double(1:5)
[1] FALSE
> is.numeric(1:5)
[1] TRUE
> is.vector(1:5) # czy wektor?
[1] TRUE
> is.atomic(1:5) # czy wektor atomowy? (bo są też wek
tory uogólnione)
[1] TRUE
```

Prealokacja wektorów

Czasem zachodzi potrzeba utworzenia wektora o zadanej długości i określonym typie elementów, który dopiero później będzie sukcesywnie wypełniany zadanymi wartościami. Wówczas najwygodniej jest wywołać np.

```
> logical(4) # tworzy wektor wartości logicznych o długości 4
[1] FALSE FALSE FALSE FALSE
> vector("logical", 4) # to samo
[1] FALSE FALSE FALSE FALSE
> integer(2)
[1] 0 0
> numeric(10)
 [1] 0 0 0 0 0 0 0 0 0 0
> character(5)
[1] "" "" "" "" ""
```

Warto zwrócić uwagę jaka wartość domyślna jest nadawana elementom poszczególnych wektorów atomowych.

8. Tworzenie obiektów nazwanych

Do danych przechowywanych w pamięci możemy odwołać się przy użyciu tzw. obiektów nazwanych. Aby utworzyć obiekt nazwany należy związać nazwę obiektu z jego wartością. Do tego celu stosuje się jeden z trzech operatorów:

- nazwa <- wartość
- wartość -> nazwa
- nazwa = wartość[3]

Nazwa nie może:

1) rozpoczynać się od cyfry, podkreślnika, cyfry poprzedzonej kropką (?make.names),
2) być słowem kluczowym (?Reserved)

W momencie utworzenia obiektu o nazwie x, nazwa x powinna się pojawić w zakładce *Environment* programu RStudio w sekcji *Values*. Wykonanie operacji przypisania nie skutkuje automatycznie wypisaniem wartości obiektu nazwanego. Można jednak wymusić takie zachowanie dodając nawiasy okrągłe.

```
> x <- 1:10
> (x <- 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

Wartość obiektu x można wypisać na konsoli na różne sposoby:

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> print(x)
 [1]  1  2  3  4  5  6  7  8  9 10
> str(x)
 int [1:10] 1 2 3 4 5 6 7 8 9 10
> cat(x)
1 2 3 4 5 6 7 8 9 10
```

Z każdą nazwą da się związać obiekt dowolnego typu , tj. typ nie jest ustalany raz na zawsze.

```
> (y <- 1:3)
[1] 1 2 3
> (y <- as.character(y))
[1] "1" "2" "3"
> (y <- TRUE)
[1] TRUE
>
```

[3] nie jest zalecany, bo ma również inne znaczenia

9. Braki danych, wartości nieskończone, nie-liczby

Braki danych (ang. missing data) w R są reprezntowane przez stałą NA (ang. not available, zobacz ?NA). Do sprawdzenia, które elementy wektora nie są znane można użyć funkcji is.na().

```
> is.na(c(7, 12, NA, -9))
[1] FALSE FALSE  TRUE FALSE
```

Większość operacji na stałej NA w wyniku da także NA, co w zasadzie jest zgodne z naszymi oczekiwaniami.

```
> 100 > NA
[1] NA
> 100 + NA
[1] NA
> as.numeric(NA)
[1] NA
> is.logical(NA)
[1] TRUE
```

Może się zdarzyć, że czasem w wyniku obliczeń uzyskuje się nie-liczbę (stała NaN, ang. not a number), inaczej wartość nieokreśloną, bądź wartość nieskończoną (stała Inf, ang. infinity).

```
> 1/0
[1] Inf
> 0/0
[1] NaN
> sqrt(-4)
[1] NaN
Komunikat ostrzegawczy:
W poleceniu 'sqrt(-4)': wyprodukowano wartości NaN
> log(0)
[1] -Inf
```

Testowanie występowania wartości specjalnych

Do sprawdzenia, które elementy wektora mają wartości nieskończone służy funkcja is.infinite(). Podobnie działa is.nan() dla wartości nieokreślonych. Częściej jednak może nas interesować is.finite().

```
> x <- c(10, NA, Inf, -Inf, NaN)
> is.finite(x)
[1]  TRUE FALSE FALSE FALSE FALSE
> is.infinite(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE  TRUE
> is.na(x)
[1] FALSE  TRUE FALSE FALSE  TRUE
```

10. Typ pusty (NULL)

NULL jest atomowym typem podstawowym, nie jest wektorem. W pewnym sensie można go postrzegać jako zbiór pusty.

```
> typeof(NULL)
[1] "NULL"
> x <- NULL
> is.null(x)
[1] TRUE
> y <- cat("Co zwraca ta funkcja?")
Co zwraca ta funkcja?
> y
NULL
```

Uwaga: NA i NULL to nie to samo.