برای آنهایی که به تازگی با زبان R آشنا می شوند، احتمالاً اولین عملگر مورد استفاده آنها عملگر تخصیص است و اینکه تخصیص یا انتساب با هر دو عملگر -> و = امکان پذیر است.

فایل های راهنمای این زبان از جمله راهنمای Google R Style Guide و حتی اساتید در تدریس این زبان استفاده از -> به جای = را پیشنهاد می کند، حتی اگر علامت مساوی در R نیز مجاز باشد تا دقیقاً همان کار اختصاص مقدار به یک متغیر را انجام دهد.

با این حال، ممکن است احساس ناراحتی کنید زیرا باید دو کاراکتر را برای نشان دادن یک عمل تایپ کنید، که با بسیاری از زبان های برنامه نویسی دیگر متفاوت است.

در نتیجه، بسیاری از کاربران می پرسند چرا باید از -> به عنوان عملگر انتساب استفاده کنیم؟

تفاوت بین -> و = عمدتا به دلیل سبک برنامه نویسی است. با این حال، کد R زیر با استفاده از علامت = نیز کار خواهد کرد:

```
my_object4 = 5
my_object4
# 5
```

در اینجا یک توضیح ساده برای تفاوت ظریف بین -> و = در R وجود دارد و ابتدا به مثال زیر توجه کنید:

```
x <- rnorm(100)
y <- 2*x + rnorm(100)
lm(formula=y~x)
```

کد بالا از هر دو نماد -> و = استفاده می کند، اما کار آنها متفاوت است!

-> در دو خط اول به عنوان عملگر انتساب استفاده می شود در حالی که = در خط سوم به عنوان عملگر انتساب عمل نمی کند بلکه عملگری است که فرمول پارامتر نامگذاری شده را برای تابع lm مشخص می کند.

به عبارت دیگر، در خط اول -> عبارت سمت راست خود یعنی rnorm(100) را ارزیابی می کند و مقدار ارزیابی شده را به متغیر سمت چپ یعنی X در محیط فعلی اختصاص می دهد. اما در خط سوم، = عبارت در سمت راست خود را (y~x) ارزیابی می کند و مقدار ارزیابی شده را به پارامتری با نام مشخص شده در سمت چپ (formula) برای یک تابع خاص تنظیم می کند.

میدانید که -> و = زمانی که به عنوان عملگر انتساب استفاده می شوند یکسان هستند. بنابراین کد فوق معادل کد زیر می باشد:

```
x = rnorm(100)
y = 2*x + rnorm(100)
lm(formula=y~x)
```

در اینجا، فقط از = استفاده شده اما برای دو هدف متفاوت: در خط اول و دوم از = به عنوان عملگر تخصیص و در خط سوم از = به عنوان یک مشخص کننده مقدار یک پارامتر استفاده شده است.

حال برای آزمایش اگر همه نمادهای = به -> تغییر کنند چه اتفاقی می‌افتد؟

```
x <- rnorm(100)
y <- 2*x + rnorm(100)
lm(formula <- y~x)
```

اگر این کد را اجرا کنید، متوجه می شوید که خروجی ها مشابه هستند. اما اگر محیط را بررسی کنید، تفاوت را مشاهده خواهید کرد: یک متغیر جدید با نام formula در محیط فعلی تعریف شده است که مقدار آن نیز برابر با y~x می باشد. پس چه اتفاقی می افتد؟

در واقع در خط سوم، دو اتفاق افتاد: اول، یک متغیر جدید با نام formula را به محیط معرفی می کند و مقدار فرمولی y~x به آن اختصاص یافته است. سپس، مقدار متغیر formula به عنوان پارامتر اول تابع lm معرفی و ارائه می شود و اگرچه در این آزمایش به معنای ارسال پارامتر به تابع، یکسان است اما همیشه اینطور نیست.

در مثال زیر، وضعیتی را نشان داده می شود که در آن -> و = به یک نتیجه منجر نمی شوند، و برخی از تفاوت های اساسی بین فلش های تخصیص و علائم مساوی را نشان داده خواهد شد.

فرض کنید که می‌خواهید میانگین یک بردار از 1 تا 5 را محاسبه کنید. می‌توانید از کد R زیر استفاده کنید:

```
mean(x = 1:5)
# 3
```

با این حال، اگر بخواهید به بردار x که در تابع میانگین استفاده کرده‌اید نگاهی بیندازید، یک پیغام خطا دریافت می‌کنید:

```
X
# Error: object 'x' not found
```

بیایید این را دقیقاً با همان کد R اما با فلش انتساب (->) به جای علامت مساوی مقایسه کنید:

```
mean(x <- 1:5)
# 3
```

خروجی تابع میانگین یکسان است. با این حال، پیکان انتساب مقادیر را در یک شی داده جدید x نیز ذخیره می کند:

```
X
# 1 2 3 4 5
```

این مثال ها تفاوت معنی داری بین -> و = را نشان می دهند. در حالی که علامت مساوی مقادیر استفاده شده را خارج از یک تابع ذخیره نمی کند، فلش تخصیص آنها را در یک شی داده جدید ذخیره می کند که می تواند خارج از تابع نیز استفاده شود.

اما برای درک تفاوت بیشتر بین این دو علامت مطلب آزمایشی دیگر و در ابتدا داده ها را بصورت زیر در نظر بگیرید:

```
x <- rnorm(100)
y <- 2*x+rnorm(100)
z <- 3*x+rnorm(100)
data <- data.frame(z,x,y)
rm(x,y,z)
```

اساساً، صرفا فقط کارهای مشابه قبلی انجام شده است با این تفاوت که همه بردارها در یک DataFrame ذخیره شده اند و آن بردارهای عددی از محیط حذف می شوند. تابع lm نیز علاوه بر یک فرمول که مشخص می شود، یک DataFrame داده را به عنوان منبع داده می پذیرد که در حالات زیر قابل تصور است.

استفاده بصورت استاندارد با استفاده از = :

```
lm(formula=z~x+y,data=data)
```

جایگزین کردن که در آن دو پارامتر نامگذاری شده دوباره مرتب می شوند:

```
lm(data=data,formula=z~x+y)
```

جایگزین کردن با تغییرات جانبی که دو متغیر جدید در محیط فعلی نیز تعریف شده است:

```
lm(formula <- z~x+y, data <- data)
```

جایگزین کردن با -> و مرتب کردن دوباره آن ها:

```
lm(data <- data, formula <- z~x+y)
```

دلیل آن دقیقاً همان چیزی اســـت که قبلاً ذکر شـــد. داده‌ها دوباره به داده‌ها اختصـــاص یافته و مقدار آن را به اولین آرگومان (formula) تابع lm فرسـتاده می شـود که فقط یک مقدار فرمول‌نویسـی شـده را می‌پذیرد. همچنین سـعی در آن شـد که مقدار z~x+y را به عنوان یک متغیر جدید اختصـاص یابد و آن را به آرگومان دوم یعنی lm (data) که صـرفا فقط مقدار نوع قاب داده (frame-typed) را می پذیرد، داده شود.

هر دو نوع پارامتری که به lm ارائه شده است اشتباه هستند، بنابراین پیام را دریافت خواهید کرد:

```
Error in as.data.frame.default(data) :
    cannot coerce class ""formula"" to a data.frame
```

از مثال‌ها و آزمایش‌های بالا، نتیجه واضـح می‌شـود که برای کاهش ابهام، باید از -> یا = به عنوان عملگر انتسـاب اسـتفاده کنید و فقط از = به عنوان مشخص‌کننده پارامتر نام‌گذاری شده برای توابع استفاده کنید.

تا اینجا فقط -> و = مقایسه شده انـد بـا ایـن حـال، روش تخصـیص دیگـری نیـز وجـود دارد کـه بایـد بـا آن نیـز آشـنا شـوید: فلش تخصیص دوگانه ->> (که به آن انتساب محدوده نیز می گویند).
کد زیر تفاوت بین -> و ->> را در R نـشان می دهد. این تفاوت عمدتاً هنگام اعمال توابع تعریف شده تو سط کاربر قابل م شاهده است. تابع کاربری زیر با یک فلش انتساب را در نظر بگیرید:

```
my_fun1 <- function() {           # User defined function with <-
  x_fun1 <- 5
}
```

حالا در R آن تابع را اجرا کنید:

```
my_fun1()                         # Execute function with <-
```

شی داده x_fun1، که مقدار 5 را در تابع به آن اختصاص داده شده بود، وجود ندارد:

```
x_fun1                            # Return output of function with <-
# Error: object 'x_fun1' not found
```

حال همین کار را با یک فلش تخصیص دوتایی انجام و سپس در R اجرا نمایید:

```
my_fun2 <- function() {           # User defined function with <<-
  x_fun2 <<- 5
}
```

```
my_fun2()                         # Execute function with <<-
```

و حالا بیایید شی داده x_fun2 را برگردانید:

```
x_fun2                            # Return output of function with <<-
# 5
```

همانطور که می بینید بر ا ساس خروجی قبلی کنـسول RStudio، انتـساب از طریق ->> شی داده را در محیط سرا سری خارج از تابع تعریف شده توسط کاربر ذخیره می کند.

با این حال و با توجه به تمام توضیحات و مثال ها، هنگام تخصیص در R باید مراقب فاصله گذاری نیز باشید. فاصله کاذب حتی می تواند منجر به پیام های خطا شود.

به عنوان مثال، کد R زیر به دلیل خالی بودن کاذب بین – و > بررسی می کند که آیا x کوچکتر از منفی پنج است یا خیر:

```
my_object1 < -5
# Error: object 'my_object1' not found
```

که در واقع شکل درست آن می تواند به صورت زیر باشد:

```
my_object2<-5
my_object2
# 5
```

با این حال، خواندن این کد دشوار است، زیرا فضای خالی، تمایز بین نمادها و اعداد مختلف را دشوار می کند.

به نظر می رسد، بهترین راه برای تخصیص در R این است که قبل و بعد از فلش انتساب یک جای خالی بگذارید:

```
my_object3 <- 5
my_object3
# 5
```

در پایان، برای خوانایی بهتر کد R، همچنان پیشنهاد می شود فقط از -> برای انتساب و = برای تعیین پارامترهای نامگذاری شده استفاده کنید و به نظر من، بسیار منطقی است که به این قراردادها پایبند باشیم تا اسکریپت هایی تولید کنیم که برای دیگر برنامه نویسان R به راحتی خوانده شوند.