Swift

به زبان ساده

تقدیم به

همه جویندگان علم

این اثر رایگان بوده و هرگونه استفاده تجاری از آن پیگرد قانونی دارد.
استفاده از مطالب آن، بدون ذکر منبع، غیراخلاقی و غیرقانونی است.

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

Swift چیست

Swift یک زبان برنامه نویسی جدید است که برای توسعه برنامه‌های iOS، Mac OS و توسعه لینوکس، توسط Chris Lattner با همکاری دیگر برنامه نویسان شرکت اپل در سال ۲۰۱۰ ایجاد شد و جایگزینی برای زبان Objective-C است، که زبان توصیه شده و محبوب‌ترین زبان، برای برنامه‌های دستگاه‌های اپل است.

نام Swift، بر گرفته از نام یکی از سریعترین پرندگان است. این زبان به عنوان یک جایگزین سریع، برای Objective-C توسعه یافت. به غیر از سرعت، Swift مزایای دیگری نسبت به Objective-C دارد، که از آن جمله می‌توان به سهولت یادگیری، ایمنی، نیاز به کد کمتر، تعاملی بودن و ... اشاره کرد.

در مقایسه با Objective-C، Swift دارای یک دستور زبان ساده‌تر است که برای مبتدیان یا کسانی که با زبان‌های برنامه نویسی دیگر کار کرده‌اند، یادگیری آن را راحت تر می‌کند. این زبان را می‌توان، تلفیقی از زبان‌های C#، Rust، Ruby، Python و طیف وسیعی از زبان‌های برنامه نویسی دیگر دانست. این بدان معنی است که برنامه نویسانی که قبلاً با زبان‌های دیگر کار کرده‌اند، مفاهیم زیادی در Swift پیدا می‌کنند، که شبیه مفاهیمی است که قبلاً در زبان‌هایی که کار کرده‌اند، با آنها مواجه شده‌اند.

Swift هنوز یک زبان برنامه نویسی جدید است اما محبوبیت و پذیرش آن نسب به سایر زبان‌های برنامه نویسی سرعت بیشتری داشته است. اکنون این زبان محبوب‌ترین زبان برای توسعه دستگاه‌های اپل است. تاکنون نسخه‌های مختلفی از این زبان ارائه شده است که در جدول زیر لیست آنها آمده است:

نسخه	تاریخ ارائه
Swift 1.0	2014-09-09
Swift 1.1	2014-10-22
Swift 1.2	2015-04-08
Swift 2.0	2015-09-21
Swift 3.0	2016-09-13
Swift 4.0	2017-09-19
Swift 4.1	2018-03-29
Swift 4.2	2018-09-17

نصب کامپایلر Swift

قبل از شروع این درس به یک نکته خیلی مهم توجه کنید که:

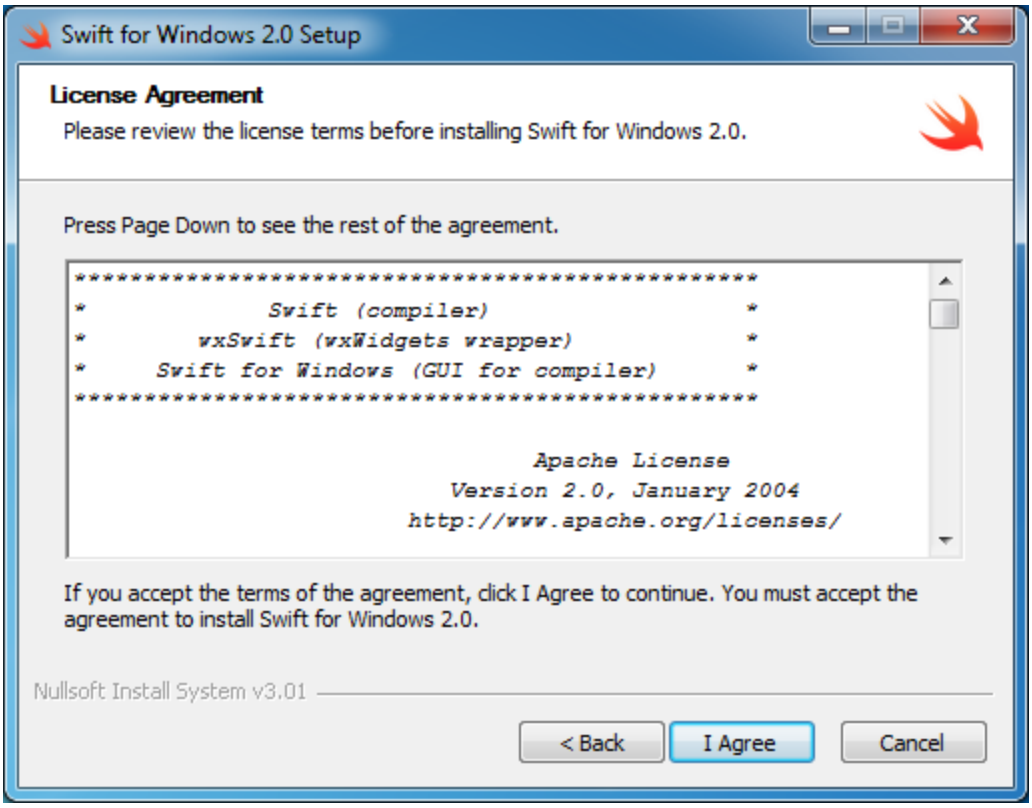
کدهای Swift در سیستم عامل Mac قابل اجرا هستند. برای کدنویسی هم به یک ویرایشگر متن به نام XCode احتیاج داریم، که باید آن را در سیستم عامل Mac نصب کنیم. پس کسانی که از سیستم عامل Mac و محیط کدنویسی XCode استفاده می‌کنند، می‌توانند این درس و درس بعد را رد کنند.

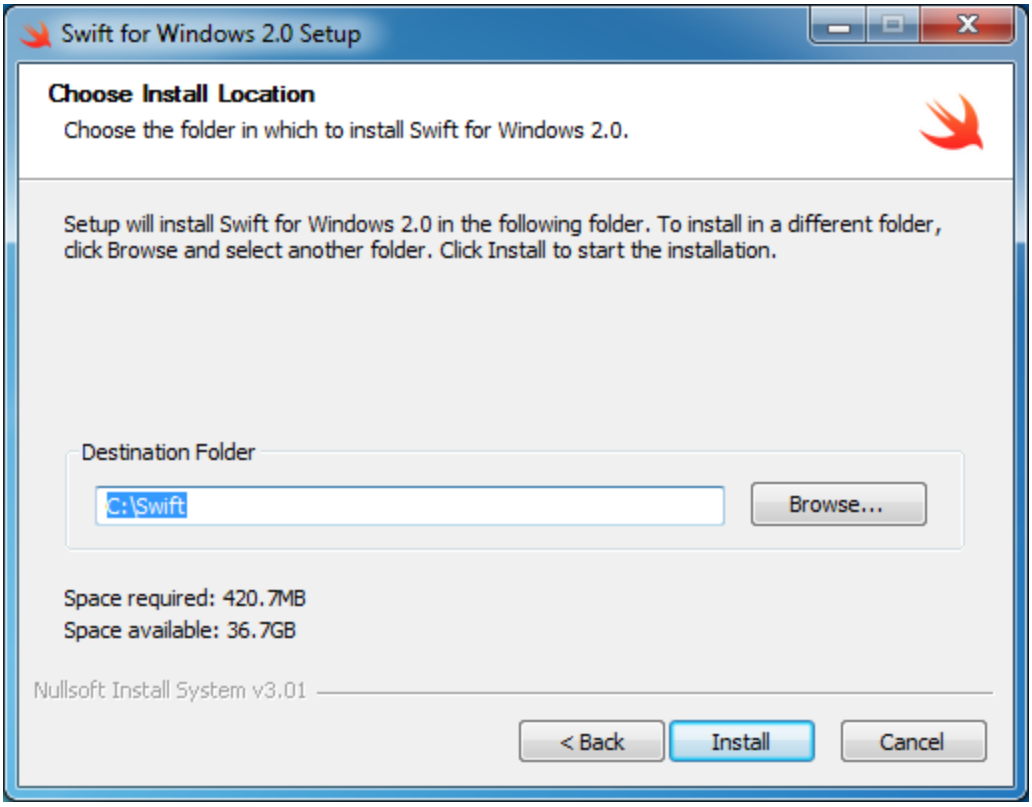
این درس برای کسانی است که به یادگیری زبان برنامه نویسی Swift علاقه دارند و می‌خواهند از سیستم عامل ویندوز و ویرایشگر متن NotePad برای یادگیری این زبان استفاده کنند. این دسته از دوستان عزیز باید کامپایلر زبان Swift را در سیستم عامل ویندوز خود نصب کرده و شروع به کدنویسی کنند. برای کدنویسی Swift، علاوه بر NotePad ویندوز، ویرایشگرهای حرفه‌ای دیگر از جمله VS Code هم وجود دارند که کار کدنویسی را راحت‌تر می‌کنند. ولی ما در این آموزش‌ها بنا را بر راحت‌ترین حالت ممکن، گذاشته‌ایم.

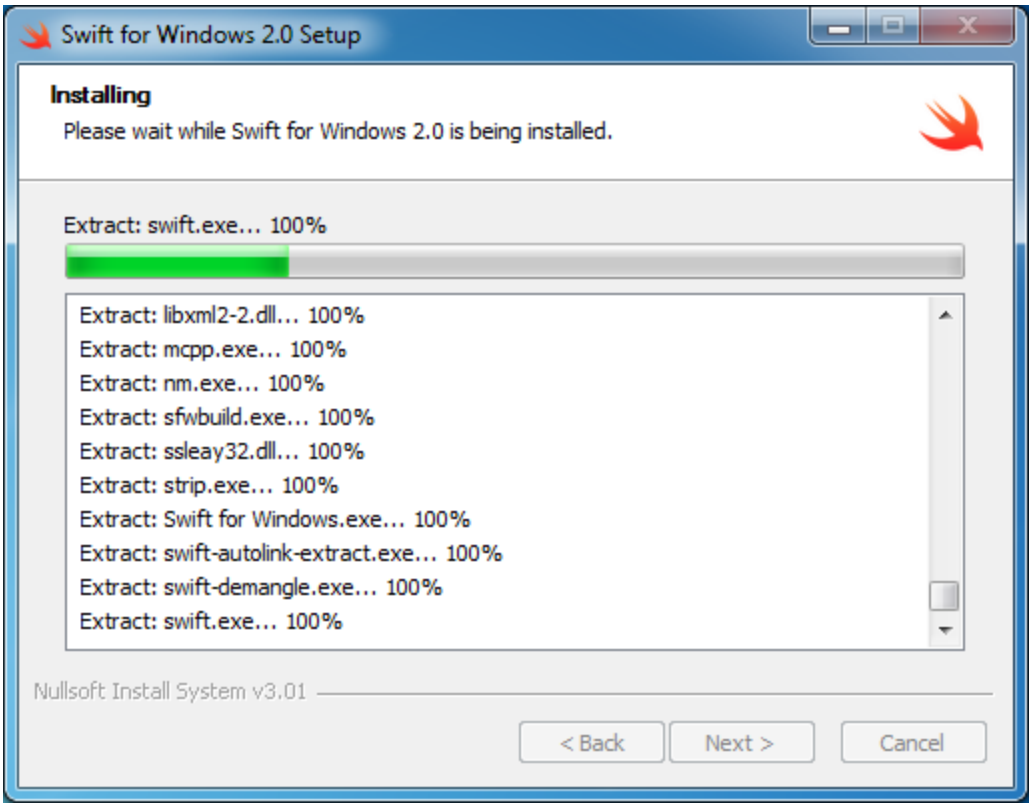
روش دیگر اجرای کدهای Swift در سیستم عامل ویندوز، نصب سیستم عامل Mac در ویندوز است که این کار توسط نرم افزاری به نام VMWare Work Station انجام می‌شود. البته این روش را توصیه نمی‌کنیم. چون نصب Mac بر روی سیستم عامل ویندوز، مستلزم داشتن یک سیستم قوی از لحاظ حافظه رم و هارد بالا و همچنین CPU های چند هسته‌ای می‌باشد. برای اجرای کدهای Swift در ویندوز، شما به کامپایلر Swift نیاز دارید که در لینک زیر می‌توانید آن را دانلود کنید:

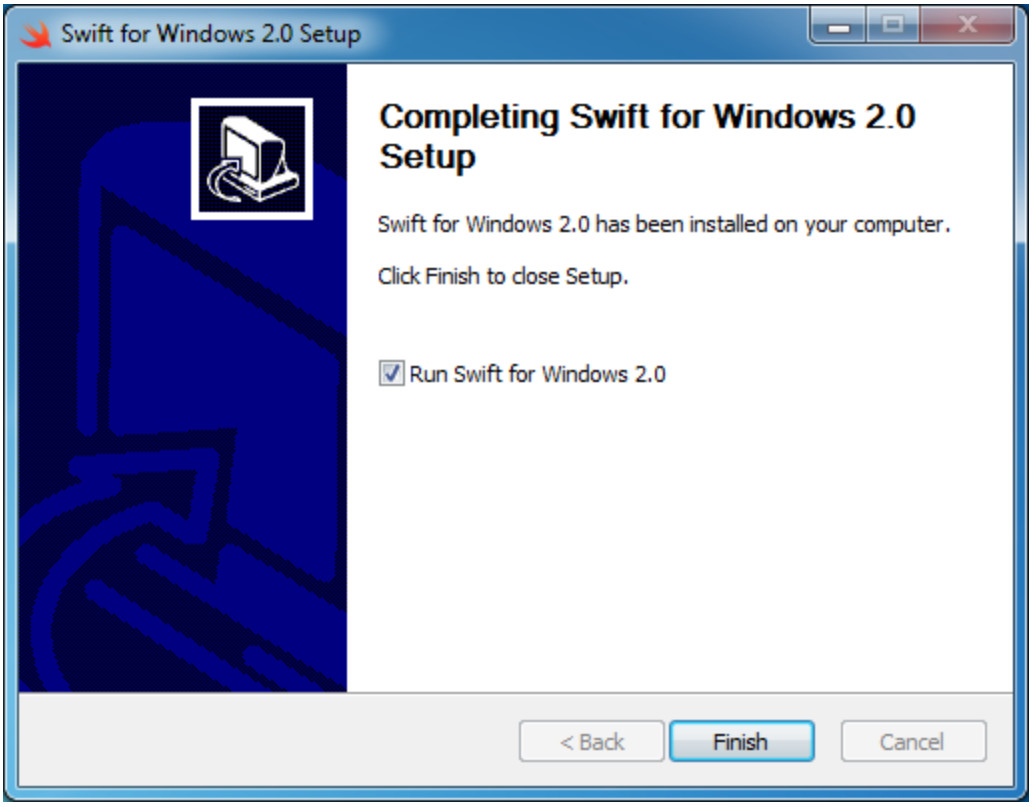
```
http://dl.w3-farsi.com/Software/Swift/SwiftForWindows-2.0.exe
```

پس از دانلود فایل بالا و دوبار کلیک بر روی آن، پنجره‌ای به صورت زیر باز می‌شود که با کلیک بر روی دکمه‌های I Agree، Install، Next و finish مراحل نصب به اتمام می‌رسد:

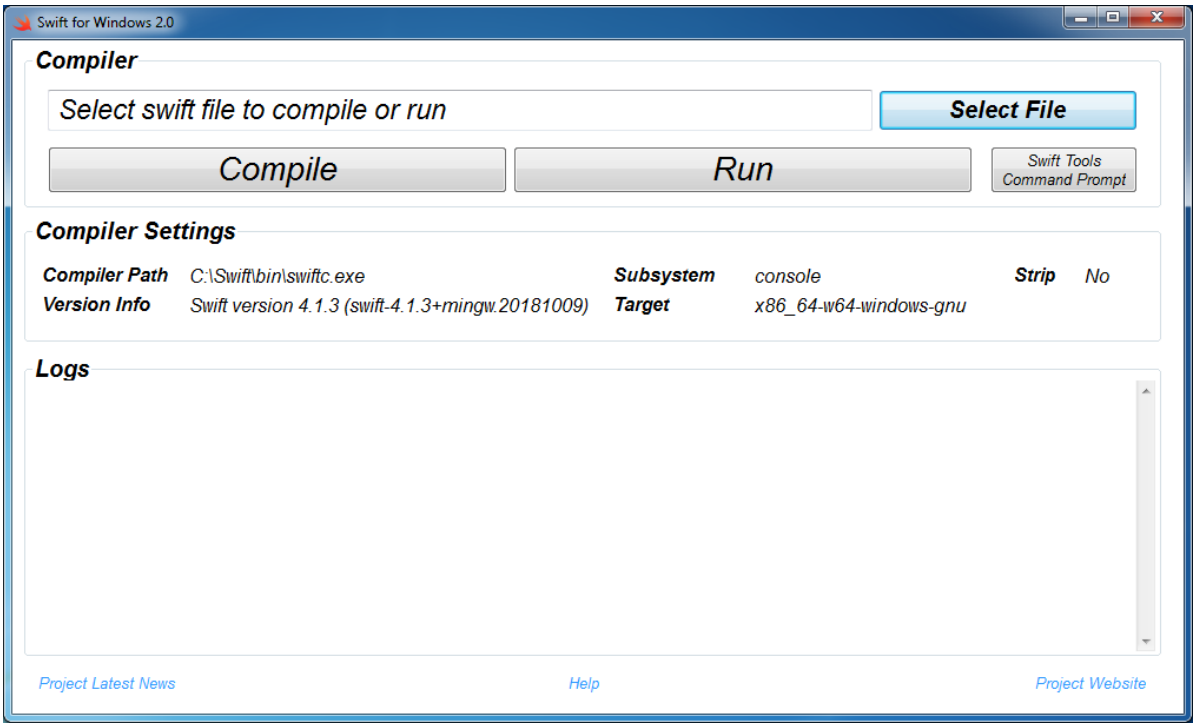

Swift for Windows 2.0 Setup
License Agreement
Please review the license terms before installing Swift for Windows 2.0.
Press Page Down to see the rest of the agreement.
****
* Swift (compiler) *
* wxSwift (wxWidgets wrapper) *
* Swift for Windows (GUI for compiler) *
****
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Swift for Windows 2.0.
Nullsoft Install System v3.01
< Back
I Agree
Cancel


Swift for Windows 2.0 Setup
Choose Install Location
Choose the folder in which to install Swift for Windows 2.0.
Setup will install Swift for Windows 2.0 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Swift
Browse...
Space required: 420.7MB
Space available: 36.7GB
Nullsoft Install System v3.01
< Back
Install
Cancel





بعد از زدن دکمه `finish` در صفحه بالا، صفحه اصلی برنامه به صورت زیر باز شده و شما می‌توانید کدنویسی را شروع کنید:

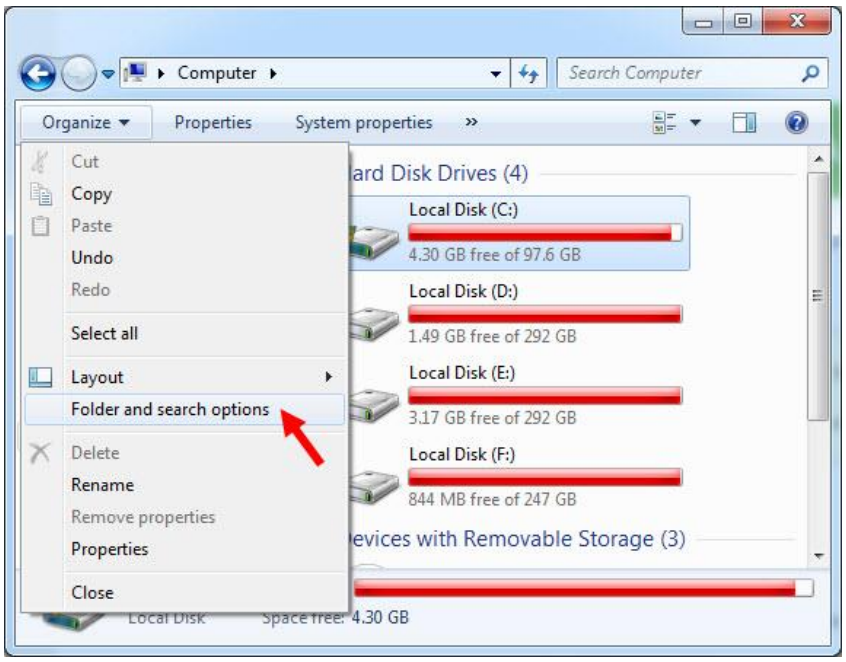


ساخت یک برنامه ساده در Swift

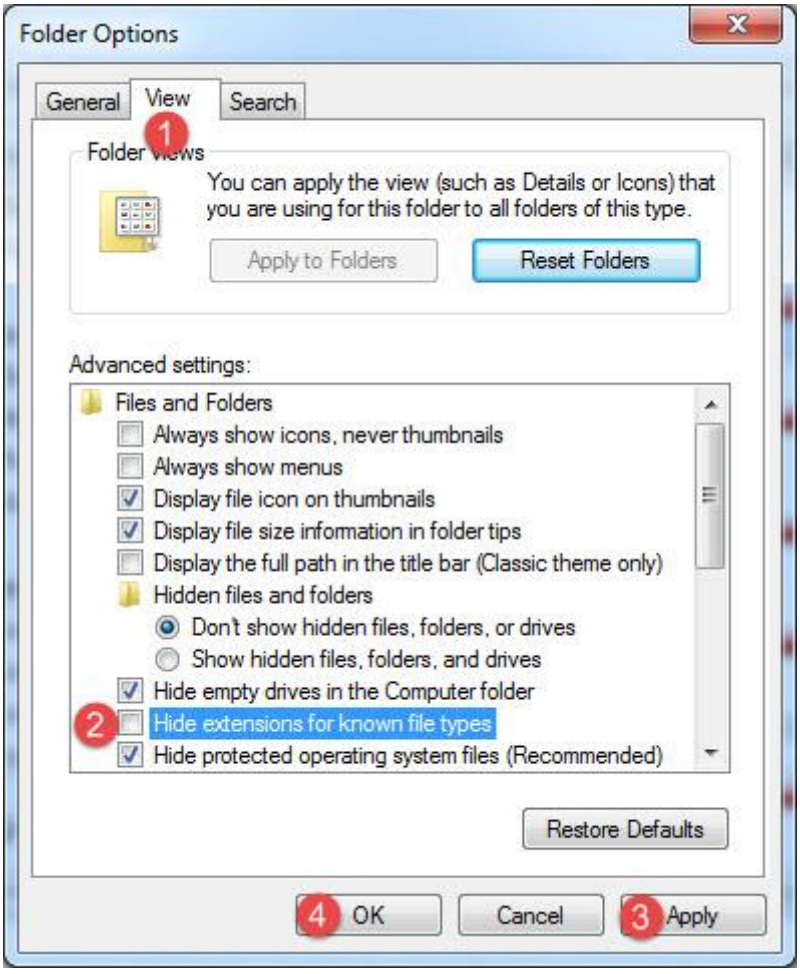
اجازه بدهید یک برنامه بسیار ساده به زبان Swift بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده Swift را توضیح دهم. قبل از ایجاد برنامه به این نکته خیلی مهم توجه کنید:

> در نوشتن این برنامه و برنامه‌های آتی، به حروف بزرگ و کوچک، توجه کنید. چون Swift به بزرگ و کوچک بودن حروف حساس است.

یکی از تنظیماتی که قبل از شروع این درس توصیه می‌کنیم که اعمال کنید این است که پسوند فایل‌ها داخل ویندوز را قابل مشاهده کنید. برای این کار به My Computer رفته و به صورت زیر از منوی Organize گزینه Folder and search options را بزنید:

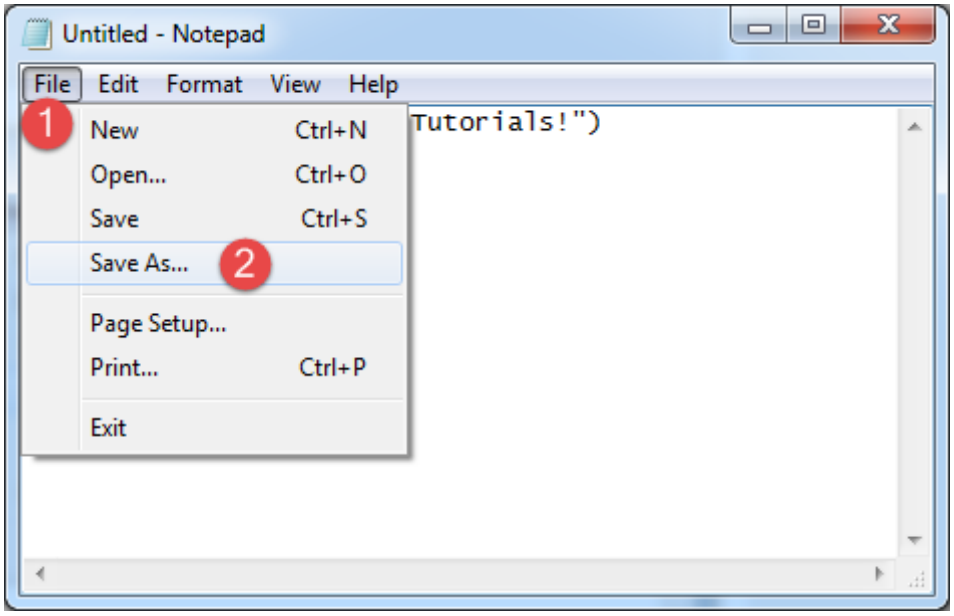


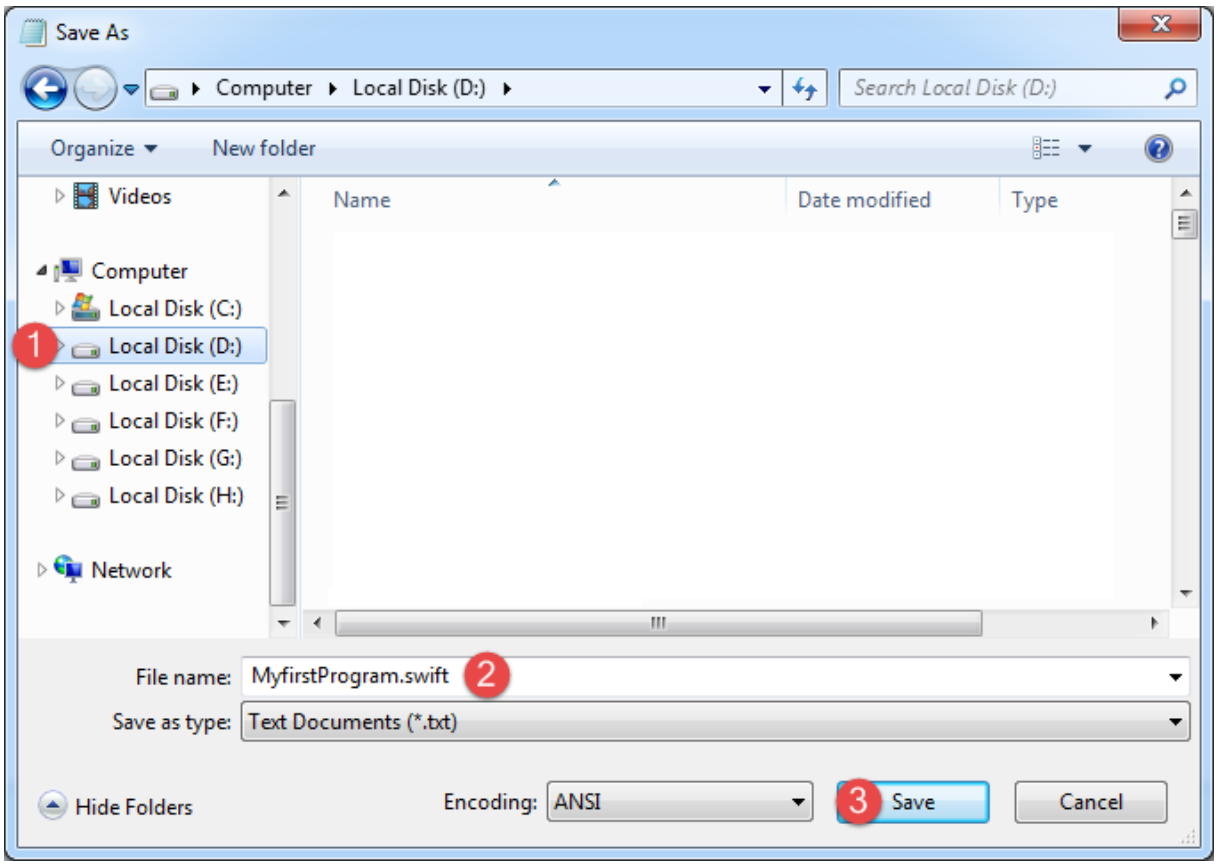
از پنجره باز شده به صورت زیر به سربرگ View رفته و تیک کنار گزینه Hide Extension for khnown file types را بردارید:

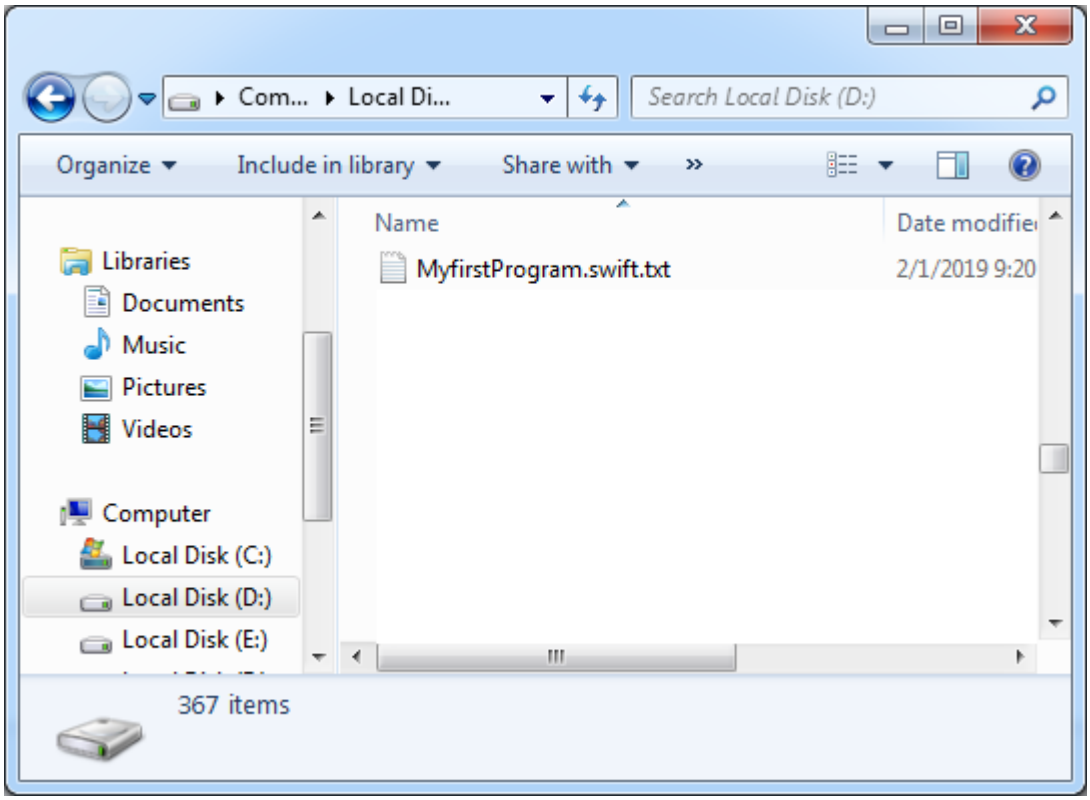


در ادامه شما را نحوه ایجاد اولین برنامه در Swift را توضیح می‌دهیم. همانطور که گفته شد، شما برای کامپایل و اجرای برنامه‌های Swift به کامپایلر این زبان نیاز دارید، که آن را در درس قبل نصب کردیم و الان فرض می‌کنیم که شما هیچ IDE یا محیط کدنویسی در اختیار ندارید و می‌خواهید یک برنامه Swift بنویسید. در این برنامه می‌خواهیم پیغام Welcome to Swift Tutorials چاپ شود. ابتدا یک ویرایشگر متن مانند Notepad را باز کرده و کدهای زیر را در داخل آن نوشته (حروف بزرگ و کوچک را رعایت کنید) و با پسوند swift ذخیره کنید :

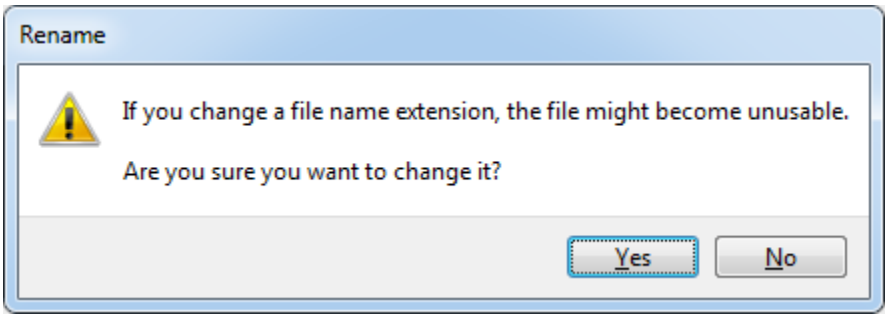
```
print("Welcome to Swift Tutorials!")
```



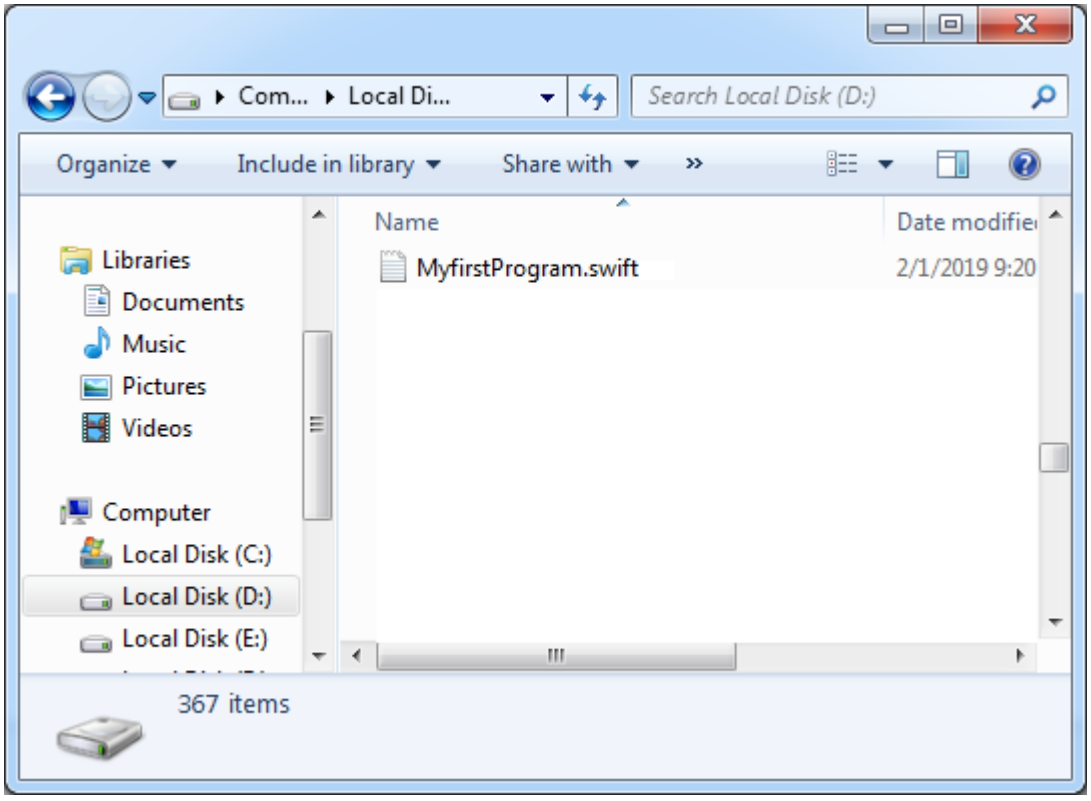

Save As
Computer ▸ Local Disk (D:) ▸
Search Local Disk (D:)
Organize
New folder
Videos
Computer
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
Local Disk (F:)
Local Disk (G:)
Local Disk (H:)
Network
Name
Date modified
Type
1
File name: MyfirstProgram.swift
2
Save as type: Text Documents (*.txt)
Hide Folders
Encoding: ANSI
3 Save
Cancel


Com... ▸ Local Di...
Search Local Disk (D:)
Organize
Include in library
Share with
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
Name
Date modifie
MyfirstProgram.swift.txt
2/1/2019 9:20
367 items

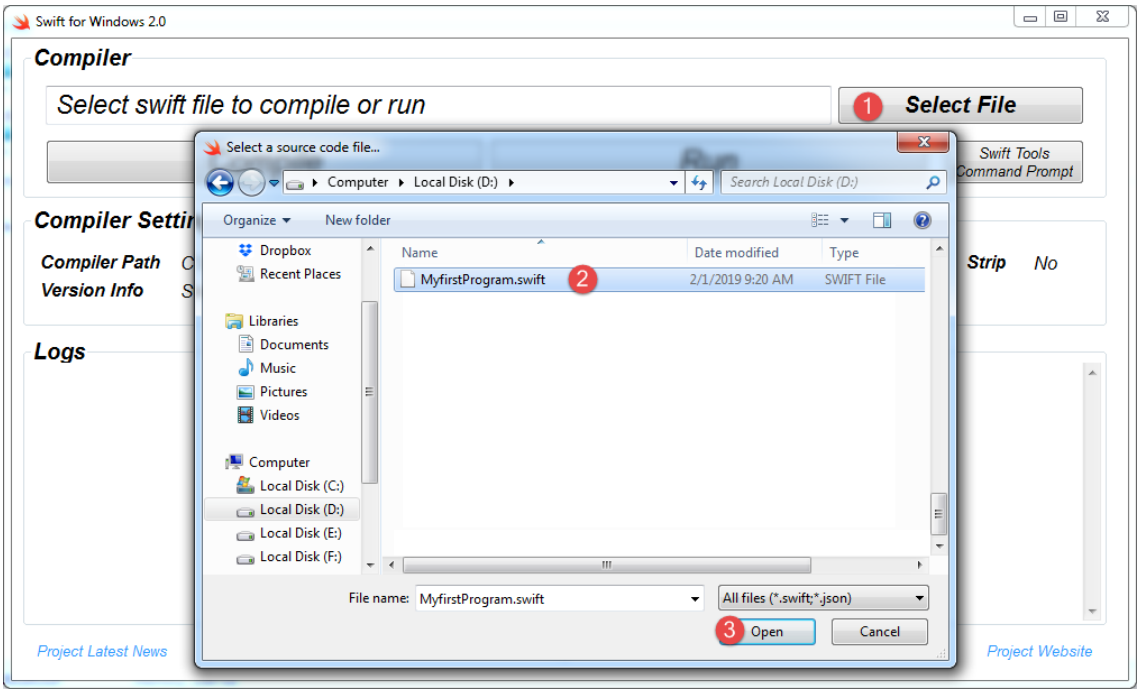
همانطور که مشاهده می‌کنید، بعد از ذخیره، فایل با پسوند MyFirstProgram.swift.txt ذخیره می‌شود که شما باید پسوند .txt آن را حذف کنید. هنگام پاک کردن پسوند، پیغامی به صورت زیر ظاهر می‌شود که شما باید بر روی گزینه Yes کلیک کنید:



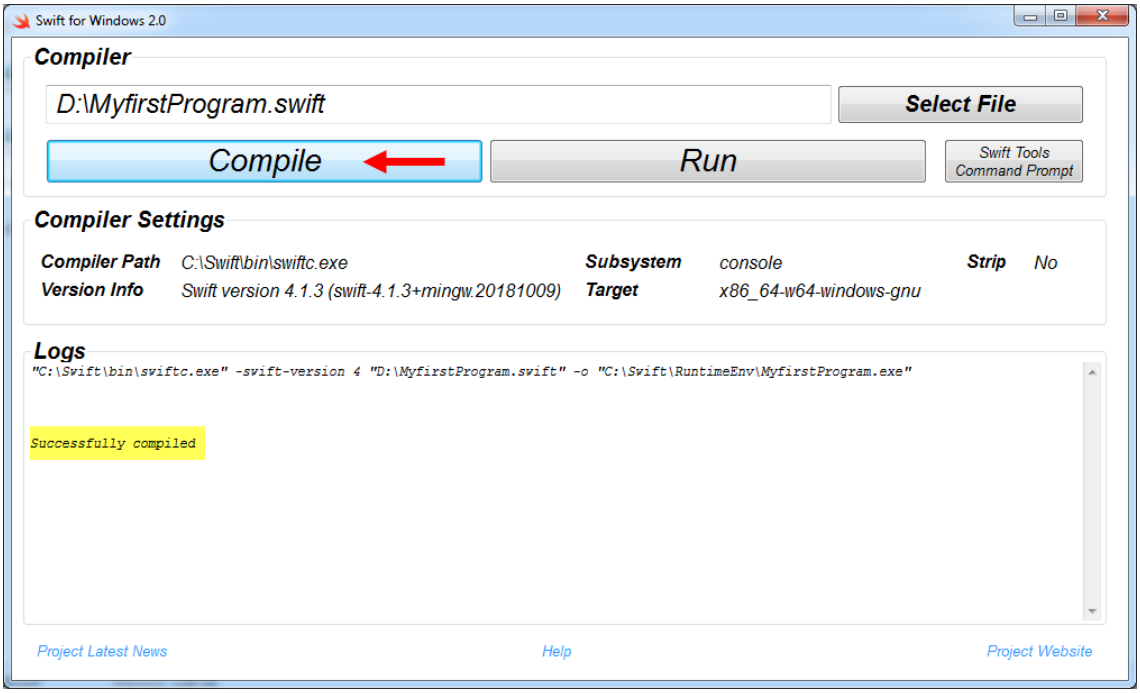
تا شکل نهایی فایل به صورت زیر در آید:



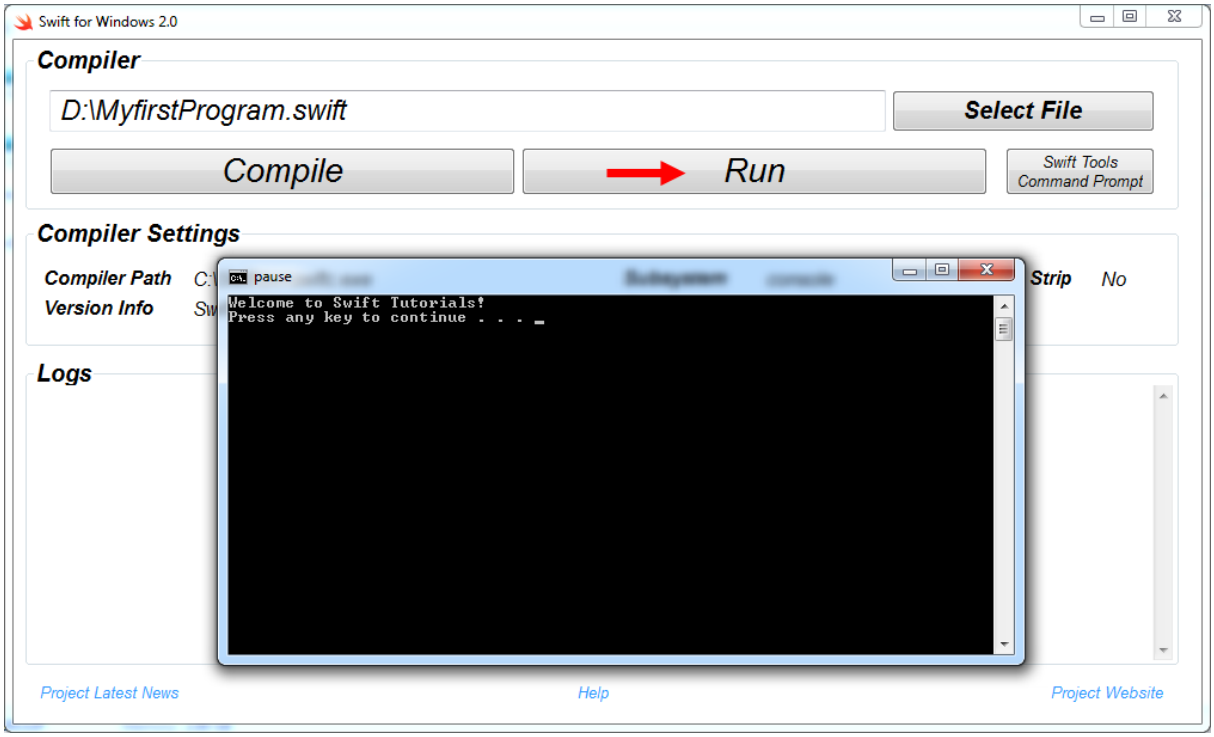
حال نوبت به اجرای برنامه می‌رسد. برنامه Swift For Windows را اجرا می‌کنیم. در صفحه زیر بر روی گزینه Select File کلیک کرده و فایل MyFirstProgram.swift را انتخاب می‌کنیم:



بعد از انتخاب فایل بر روی دکمه Compile کلیک کرده و صبر می‌کنیم تا پیغام Successfully compiled نمایش داده شود:



در نهایت بعد از کلیک بر روی دکمه Run برنامه اجرا شده و پیغام نمایش Welcome to Swift Tutorials! داده می‌شود:



مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در Swift بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. Swift دارای توابع از پیش تعریف شده‌ای است که هر کدام برای مقاصد خاصی به کار می‌روند. هر چند که در آینده در مورد توابع بیشتر توضیح می‌دهیم، ولی در همین حد به توضیح تابع بسنده می‌کنیم که توابع مجموعه‌ای از کدها هستند که دارای یک نام بوده و در جلوی نام آنها علامت () قرار می‌گیرد. یکی از این توابع، تابع print() است.

از تابع print() برای چاپ یک رشته استفاده می‌شود. یک رشته گروهی از کاراکترها است، که به وسیله دابل کوتیشن (") محصور شده است. مانند: "Welcome to Swift Tutorials!" یک کاراکتر می‌تواند یک حرف، عدد، علامت یا ... باشد. در کل مثال بالا نحوه استفاده از تابع print() است. توضیحات بیشتر در درس‌های آینده آمده است. Swift فضاهای خالی را نادیده می‌گیرد. مثلاً از کد زیر اشکال نمی‌گیرد:

```
print(
    "Welcome to Swift Tutorials!")
```

همیشه به یاد داشته باشید که Swift به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در Swiftبا هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیح خواهیم داد. مثلاً کدهای زیر با خطا مواجه می‌شوند و اجرا نمی‌شوند:

```
Print("Welcome to Swift Tutorials!")
PRINT("Welcome to Swift Tutorials!")
PrinT("Welcome to Swift Tutorials!")
```

تغییر در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطا است:

```
print("Welcome to Swift tutorials!")
```

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آنها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد :

```
print("Hello\nWorld!")
```

```
Hello
World
```

مشاهده کردید که کامپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برده و بقیه رشته را در خط بعد نمایش می‌دهد. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد :

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
Form Feed	\f	چاپ کوتیشن	\’
خط جدید	\n	چاپ دابل کوتیشن	\”
سر سطر رفتن	\r	چاپ بک اسلش	\\
حرکت به صورت افقی	\t	چاپ فضای خالی	\0
حرکت به صورت عمودی	\v	صدای بیپ	\a
چاپ کاراکتر یونیکد	\u	حرکت به عقب	\b

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش (\) باید از (\\) استفاده کنیم :

```
print("We can print a \\ by using the \\\\ escape sequence.")
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \\، نشان دادن مسیر یک فایل در Mac است :

```
print("C:\\Program Files\\Some Directory\\SomeFile.txt")
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن (“) برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از ”\ استفاده می‌کنیم :

```
print("I said, \"Motivate yourself!\".")
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از ’\ استفاده می‌کنیم :

```
print("The programmer\'s heaven.")
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از t\ استفاده می‌شود :

```
print("Left\tRight")
```

```
Left    Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی r\ بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند :

```
print("Mitten\rK")
```

```
Kitten
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی r\ آمده است. کاراکتر کنترلی حرف K را به ابتدای سطر برده و جایگزین حرف M می‌کند.

برای چاپ کاراکترهای یونیکد می‌توان از u\ استفاده کرد. برای استفاده از u\، مقدار در مبنای ۱۶ کاراکتر را درست بعد از علامت u\ و در داخل دو علامت {} قرار می‌دهیم. برای مثال اگر بخواهیم علامت کپی رایت (©) را چاپ کنیم، باید بعد از علامت u\ مقدار A9۰۰ را قرار دهیم مانند :

```
print("\u{00A9}")
```

```
©
```

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید :

http://www.ascii.cl/htmlcodes.htm

اگر کامپایلر به یک کاراکتر کنترلی غیر مجاز برخورد کند، برنامه پیغام خطا می‌دهد. بیشترین خطا زمانی اتفاق می افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند. برای دریافت اطلاعات بیشتر در مورد کاراکترهای کنترلی به لینک زیر مراجعه کنید :

https://msdn.microsoft.com/en-us/library/h21280bw.aspx

از سایر کتاب های یونس ابراهیمی در سایت W3-farsi.com دیدن فرمایید.

توضیحات

وقتی که کدی تایپ می کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در Swift (و بیشتر زبانهای برنامه نویسی) می توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کامپایلر نادیده گرفته می شوند و به عنوان بخشی از کد محسوب نمی شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می خواهید در مورد یک کد خاص، توضیح بدهید، می توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می شود. در برنامه زیر نقش توضیحات نشان داده شده است:

```
//This line will print the message hello world
print("Hello World!")
```

خط اول کد بالا یک توضیح درباره خط دوم است که به کاربر اعلام می کند که وظیفه خط دوم چیست ؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط اول در خروجی نمایش داده نمی شود چون کامپایلر توضیحات را نادیده می گیرد. همانطور که مشاهده می کنید برای درج توضیحات در Swift از علامت // استفاده می شود. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشته‌ای که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود.

```
/*This line will print
  the message hello world*/
print("Hello World!")
```

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است.

متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند، داریم. این مکان، همان متغیر است.

برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد، مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعی مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز

پاک می‌شود. قبلاً ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- نام متغیر باید با یکی از حروف الفبا (a-z or A-Z) یا علامت _ شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند . , $, ^, ?, # باشد.
- نمی‌توان از کلمات رزرو شده در Swift برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در Swift دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آیند.

دو متغیر با نامهای myNumber و MyNumber دو متغیر مختلف محسوب می‌شوند چون یکی از آنها با حرف کوچک m و دیگری با حرف بزرگ M شروع می‌شود. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. در درس بعد در مورد انواع داده ها در Swift توضیح می دهیم. لیست کلمات کلیدی Swift، که نباید از آنها در نامگذاری متغیرها استفاده کرد در زیر آمده است:

Class	deinit	Enum	extension	Func	import	Init
operator	private	protocol	public	static	struct	subscript
break	case	continue	default	do	else	for
return	switch	where	while	as	false	is
dynamicType	super	true	_COLUMN_	Let	in	_FILE_
internal	typealias	if	nil	var	self	unowned
FUNCTION	_LINE_	associativity	convenience	dynamic	didSet	precedence
final	get	infix	inout	right	set	type
lazy	left	mutating	none	weak	willSet	prefix
nonmutating	optional	override	postfix	Protocol	required	

انواع داده

متغیرها در Swift می‌توانند انواع مختلف داده را در خود ذخیره کنند. این داده‌ها دارای مجموعه مشخصی از مقادیر شامل اعداد، کاراکترها و یا مقادیر بولی هستند. در جدول زیر انواع داده و محدوده آنها آمده است :

نوع		مقدار فضایی که اشغال می کند	دامنه
Int	Int8	1byte	-۱۲۷ تا ۱۲۸
	UInt8	1byte	۰ تا ۲۵۵
	Int32	4bytes	-۲۱۴۷۴۸۳۶۴۸ تا ۲۱۴۷۴۸۳۶۴۷
	UInt32	4bytes	۰ تا ۴۲۹۴۹۶۷۲۹۵
	Int64	8bytes	-۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸ تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۷
	UInt64	8bytes	۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵
Float		4bytes	-۳٫۴۰۲۸۲۳E۳۸ تا ۳٫۴۰۲۸۲۳E۳۸
Double		8bytes	-۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸ تا ۱٫۷۹۷۶۹۳۱۳۴۸۶۲۳۲E۳۰۸

برای به خاطر سپردن دو نوع Float و Double در جدول بالا، باید از نماد علمی استفاده شود. انواع ساده دیگری که از آنها برای ذخیره داده‌های غیر عددی و پیچیده تر استفاده می شود و در جدول زیر نمایش داده شده‌اند :

نوع	توضیح
Bool	برای ذخیره مقدار true یا false به کار می‌رود.
String	برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود.
Character	برای ذخیره کاراکترهای یونیکد به کار می‌رود.
Optional	یک متغیر است که می‌تواند دارای مقدار باشد یا نه.
Tuples	برای ذخیره چندین مقدار در یک شکل واحد استفاده می‌شود.

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهی متغیرها نمایش داده شده است :

```
1 //Declare variables
2 var num1    :Int
```

```
var num2    :Int
var num3    :Float
var num4    :Float
var boolVal :Bool
var myChar  :String

//Assign values to variables
num1    = 1
num2    = 2
num3    = 3.54
num4    = 4.12
boolVal = true
myChar  = "R"

//Show the values of the variables
print("num1    = \(num1   )")
print("num3    = \(num3   )")
print("num4    = \(num4   )")
print("boolVal = \(boolVal)")
print("num2    = \(num2   )")
print("myChar  = \(myChar )")
```

```
num1    =  1
num3    =  3.54
num4    =  4.12
boolVal =  true
num2    =  2
myChar  =  "R"
```

تعریف متغیر

برای تعریف متغیر از دو کلمه var به صورت زیر استفاده می‌شود:

```
var identifier :type
```

var یک کلمه کلیدی، identifier نام و type نوع متغیر است. در این روش ما صراحتاً نوع متغیر را ذکر می‌کنیم. توجه کنید که قبل از نوع متغیر باید علامت : را قرار دهید. یک روش دیگر هم برای تعریف متغیر وجود دارد و آن این است که نوع متغیر را صراحتا اعلام نکنیم :

```
var identifier = value
```

در روش بالا، باید فوراً بعد از علامت مساوی مقدار متغیر را هم بنویسیم، تا کامپایلر به صورت خودکار و با توجه به مقدار نوع متغیر را تشخیص دهد ولی در روش اول همانطور که در کد ابتدای درس می‌بینیم، ما اول متغیرها را در خطوط ۲-۷ تعریف و سپس در خطوط ۱۰-۱۵ مقداردهی کرده‌ایم. پس خطوط ۸-۲۱ کد بالا را می‌توان به صورت زیر هم خلاصه کرد:

```
num1    = 1
num2    = 2
num3    = 3.54
num4    = 4.12
boolVal = true
myChar  = "R"
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب نوع و نام برای متغیر (خطوط ۷-۲) ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر (خطوط ۱۵-۱۰). برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد :

```
var identifier1, identifier2, ... indentifierN: data_type
```

مثال

```
var num1, num2, num3, num4, num5: Int
var message1, message2, message3: String
```

در مثال بالا ۵ متغیر از نوع صحیح و ۳ متغیر از نوع رشته تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما (,) باشد. روش دیگر برای تعریف و مقداردهی چندین متغیر در یک خط، چه از یک نوع باشند و چه نباشند، به صورت زیر است:

```
var x = 0.0, y = 10, z = "This is a string"
```

در این روش نوع آنها را مشخص نمی کنیم تا کامپایلر نوع متغیرها را به صورت خودکار تشخیص دهد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.
- نمی‌توان از کاراکترهای خاص مانند # ،% ،& یا عدد برای شروع نام متغیر استفاده کرد مانند 2numbers.
- نام متغیر نباید دارای فاصله باشد. برای نام‌های چند حرفی می‌توان به جای فاصله از علامت زیرخط یا _ استفاده کرد.

نامهای مجاز :

```
num1  myNumber  studentCount  total       first_name    _minimum
num2  myChar    average       amountDue   last_name     _maximum
name  counter   sum           isLeapYear  color_of_car  _age
```

نامهای غیر مجاز :

```
123        #numbers#  #ofstudents  1abc2
123abc     $money     first name   ty.np
my number  this&that  last name    1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آن‌ها خواهید شد. یکی از روشهای نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه‌ای به کار می‌رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می‌شود مانند :myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید، بعد از اولین کلمه، حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

به خطوط ۲۳-۱۸ کد بالا، توجه کنید. نام متغیرها در این خطوط در داخل ()\ نوشته شده است. وجود نام متغیرها در بین دو علامت پرانتز و بعد از علامت \ بدین معناست که ما مقدار متغیر را می خواهیم. برای درک بهتر خطوط ۲۳-۱۸ کد بالا را به صورت زیر نوشته و برنامه را اجرا کنید:

```
print("num1    = \(num1   )")
print("num3    = \(num3   )")
print("num4    = \(num4   )")
print("boolVal = \(boolVal)")
print("num2    = \(num2   )")
print("myChar  = \(myChar )")
```

با اجرای برنامه، نام متغیرها چاپ می شود نه مقدار آنها :

```
num1    = num1
num3    = num3
num4    = num4
boolVal = boolVal
num2    = num2
myChar  = myChar
```

پس اگر مقدار یک متغیر را می خواهید چاپ کنید، باید نام متغیر را در داخل ()\ بنویسید، مانند \(num1).

ثابت

ثابت‌ها، انواعی از متغیرها هستند که مقدار آنها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آنها فراموش شود در برنامه خطا به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی let استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با علامت زیر خط شروع می کنند تا تشخیص آنها در برنامه راحت باشد (ولی این یک قرارداد است و شما می توانید طبق قوانین نامگذاری متغیرها یک نام برای ثابت تعریف کنید). نحوه تعریف ثابت در زیر آمده است :

```
let identifier :data_type = initial_value
```

مثال :

```
let _number :Int = 1

 _number = 10 //error: cannot assign to value: ' _number' is a 'let' constant
```

در این مثال می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطا مواجه می‌کند. ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت‌ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی‌کنند بهتر است که آنها را به صورت ثابت تعریف کنید. این کار هر چند کوچک کیفیت برنامه شما را بالا می‌برد.

تبدیل انواع داده

در زبان Swift امکان تبدیل یک نوع به نوع دیگر وجود دارد که اصطلاحاً به آن Type Casting گفته می‌شود. Swift دارای مجموعه‌ای از کلاس‌های از پیش تعریف شده است، که می‌توانند مقادیر را از یک نوع بـه نوع دیگر تبدیل کنند. در Swift بر خلاف بسیاری از زبان‌های برنامه نویسی، تبدیل ضمنی وجود ندارد. یعنی یک نوع داده به طور خودکار به نوع دیگر تبدیل نمی‌شود. به کد زیر توجه کنید :

```
var number1: Double = 5.25
```

```
var number2: Int    = number1
```

در کد بالا، ما یک متغیر از نوع Double به نام number1 تعریف کرده و آن را با استفاده از علامت = در داخل یک متغیر از نوع Int به نام number2 قرار داده‌ایم. با این کار با از کامپایلر خواسته‌ایم که یک نوع Double را به نوع Int تبدیل کند. ولی کامپایلر Swift این کار را انجام نمی‌دهد و برنامه با خطا مواجه می‌شود. برای انجام همچین کاری باید از تبدیل صریح استفاده کنیم که یک نوع تبدیل اجباری بوده و کامپایلر را وادار به انجام می‌کند و نحوه انجام آن به صورت زیر است :

```
var variableB: datatypeB = datatypeB(variableA)
```

با توجه به کد بالا، اگر بخواهیم یک نوع Double را به یک نوع Int تبدیل کنیم، باید به صورت زیر عمل نماییم:

```
var number1: Double = 5.25
var number2: Int    = Int(number1)

print(number2)
```

```
5
```

حال اگر برنامه را اجرا کنید با خطا مواجه نخواهید شد. به این نکته توجه کنید که در هنگام استفاده از تبدیل صریح ممکن است، مقادیر اصلی تغییر کنند. برای مثال، وقتی که در کد بالا، یک عدد با ممیز اعشار مثلاً از نوع Double را به یک نوع Int تبدیل می‌کنیم، مقدار اعداد بعد از ممیز از بین می‌روند. خروجی کد بالا عدد ۵ است چون نوع داده‌ای Int نمی‌تواند مقدار اعشار بگیرد. حالت دیگر را تصور کنید. اگر شما بخواهید یک متغیر را که دارای مقداری بیشتر از محدوده متغیر مقصد هست تبدیل کنید، چه اتفاقی می افتد؟ مانند تبدیل زیر که می‌خواهیم متغیر number1 را که دارای مقدار ۳۰۰ است را به نوع بایت تبدیل کنیم که محدود اعداد بین ۲۵۵-۰ را پوشش می‌دهد:

```
var number1: UInt32 = 300
var number2: UInt8 = UInt8(number1)

print(number2)
```

```
Fatal error: Not enough bits to represent a signed value
```

برنامه با خطا مواجه می‌شود. دلیل این امر این است که، UInt8 فقط می‌تواند شامل اعداد ۰ تا ۲۵۵ باشد و نمی‌تواند مقدار ۳۰۰ را در خود ذخیره کند. علت خطا به تعداد بیتها بستگی دارد. یک UInt8 دارای ۸ بیت است درحالی که UInt32 دارای ۳۲ بیت است.

```
300 = 00000000000000000000000100101100
255 =                         11111111
```

خروجی بالا نشان می‌دهد که بیشترین مقدار UInt8 که عدد ۲۵۵ است، می‌تواند فقط شامل ۸ بیت باشد (11111111) بنابراین فقط ۸ بیت اول مقدار UInt32 به متغیر UInt8 انتقال می‌یابد و چون بقیه بیت‌ها قابل انتقال نیستند با خطای Not enough bits to represent a signed value مواجه می‌شویم. می‌توان با استفاده از Type Casting یا تبدیل صریح یک نوع رشته‌ای را هم به انواع عددی تبدیل کرد. به کد زیر توجه کنید :

```
var stringValue = "300"
var numberValue = Int(stringValue)!

print(numberValue)
```

```
300
```

به این نکته توجه کنید که در تبدیل یک نوع رشته‌ای به عددی باید بعد از پرانتز بسته از علامت ! استفاده کنید. حال چطور بفهمیم که واقعاً تبدیل درست انجام شده و numberValue از نوع Int است؟ برای این منظور از دستور type(of:) استفاده می‌کنیم. در ادامه کدهای بالا، دستور زیر را بنویسید:

```
print(type(of: numberValue))
```

برنامه را بار دیگر کامپایل و اجرا و نتیجه را مشاهده کنید:

```
300
Int
```

برای دریافت نسخه کامل این کتاب به سایت w3-farsi.com مراجعه فرمایید.

عبارات و عملگرها

ابتدا با دو کلمه آشنا شوید :

- عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.
- عملوند: مقادیری که عملگرها بر روی آنها عملی انجام می‌دهند.

مثلاً X+Y: یک عبارت است که در آن X و Y عملوند و علامت + عملگر به حساب می‌آیند.

زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. Swift دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه‌ای، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. انواع مختلف عملگر که در این بخش مورد بحث قرار می‌گیرند، عبارتند از :

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه‌ای
- عملگرهای منطقی
- عملگرهای بیتی
- عملگرهای محدوده
- عملگرهای متفرقه

عملگرهای ریاضی

Swift از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی Swift را نشان می‌دهد :

عملگر	مثال	نتیجه
+	var1 = var2 + var3	Var1 برابر است با حاصل جمع var2 و var3
-	var1 = var2 - var3	Var1 برابر است با حاصل تفریق var2 و var3
*	var1 = var2 * var3	Var1 برابر است با حاصلضرب var2 در var3
/	var1 = var2 / var3	Var1 برابر است با حاصل تقسیم var2 بر var3
%	var1 = var2 % var3	Var1 برابر است با باقیمانده تقسیم var2 و var3

استفاده از عملگرهای ریاضی برای نوع رشته‌ای نتیجه متفاوتی دارد. اگر از عملگر + برای رشته‌ها استفاده کنیم دو رشته را با هم ترکیب کرده و به هم می‌چسباند. حال می‌توانیم با ایجاد یک برنامه نحوه عملکرد عملگرهای ریاضی در Swift را یاد بگیریم :

```
//Variable declarations
var num1:Int
var num2:Int
var msg1:String
var msg2:String

//Assign
num1 = 5
num2 = 3

//Demonstrate use of mathematical operators
print("The sum        of \(num1) and \(num2) is \(num1 + num2).")
print("The difference of \(num1) and \(num2) is \(num1 - num2).")
print("The product    of \(num1) and \(num2) is \(num1 * num2).")
print("The quotient   of \(num1) and \(num2) is \(num1 / num2).")
print("The remainder  of \(num1) divided by \(num2) is \(num1 % num2).")

//Demonstrate concatenation on strings using the + operator
msg1 = "Hello "
msg2 = "World!"
print(msg1 + msg2)
```

```
The sum        of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product    of 5 and 3 is 15.
The quotient   of 5 and 3 is 1.
The remainder  of 5 divided by 3 is 2.
Hello World!
```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از متد print() برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در خط ۲۱ مشاهده می‌کنید که دو رشته به وسیله عملگر + به هم متصل شده‌اند. نتیجه استفاده از عملگر + برای چسباندن دو کلمه "Hello " و "World!" رشته "Hello World!" خواهد بود. به فاصله‌های خالی بعد از اولین کلمه توجه کنید اگر آنها را حذف کنید از خروجی برنامه نیز حذف می‌شوند.

عملگرهای تخصیصی (جایگزینی)

نوع دیگر از عملگرهای Swift عملگرهای جایگزینی نام دارند. این عملگرها مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در Swift را نشان می‌دهد:

عملگر	مثال	نتیجه
=	var1 = var2	مقدار var1 برابر است با مقدار var2
+=	var1 += var2	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	var1 -= var2	مقدار var1 برابر است با حاصل تفریق var1 و var2
*=	var1 *= var2	مقدار var1 برابر است با حاصل ضرب var1 در var2
/=	var1 /= var2	مقدار var1 برابر است با حاصل تقسیم var1 بر var2
%=	var1 %= var2	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2

از عملگر =+ برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً شکل اصلی کد var1 += var2 به صورت var1 = var1 + var2 می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آنها را بر متغیرها نشان می‌دهد.

```
1  var number :Int
2
3  print("Assigning 10 to number...")
4  number = 10
5  print("Number = ", number)
6
7  print("Adding 10 to number...")
8  number += 10
9  print("Number = ", number)
10
11 print("Subtracting 10 from number...")
12 number -= 10
13 print("Number = ", number)
```

```
Assigning 10 to number...
```

```
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر = به آن اختصاص داده شده است. سپس به آن با استفاده از عملگر =+ مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر =- عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه ای

از عملگرهای مقایسه‌ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار true و اگر نتیجه مقایسه اشتباه باشد مقدار false را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه‌ای در Swift را نشان می‌دهد:

عملگر	مثال	نتیجه
==	var1 = var2 == var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر باشد در غیر اینصورت false است
!=	var1 = var2 != var3	var1 در صورتی true است که مقدار var2 با مقدار var3 برابر نباشد در غیر اینصورت false است
<	var1 = var2 < var3	var1 در صورتی true است که مقدار var2 کوچک‌تر از var3 مقدار باشد در غیر اینصورت false است
>	var1 = var2 > var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر ازمقدار var3 باشد در غیر اینصورت false است
<=	var1 = var2 <= var3	var1 در صورتی true است که مقدار var2 کوچک‌تر یا مساوی مقدار var3 باشد در غیر اینصورت false است
>=	var1 = var2 >= var3	var1 در صورتی true است که مقدار var2 بزرگ‌تر یا مساوی var3 مقدار باشد در غیر اینصورت false است

برنامه زیر نحوه عملکرد این عملگرها را نشان می‌دهد :

```
1  var num1 :Int = 10
```

```swift
var num2 :Int = 5

print("num1 == num2 : \(num1 == num2)")
print("num1 != num2 : \(num1 != num2)")
print("num1 <  num2 : \(num1 < num2 )")
print("num1 >  num2 : \(num1 > num2 )")
print("num1 <= num2 : \(num1 <= num2)")
print("num1 >= num2 : \(num1 >= num2)")
```

```
num1 == num2 : false
num1 != num2 : true
num1 <  num2 : false
num1 >  num2 : true
num1 <= num2 : false
num1 >= num2 : true
```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آنها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه‌ای آنها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند x = y مقدار y را در به x اختصاص می‌دهد. عملگر == عملگر مقایسه‌ای است که دو مقدار را با هم مقایسه می‌کند مانند x==y و اینطور خوانده می‌شود x برابر است با y.

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آنها نیز یک مقدار بولی است. از این عملگرها اغلب برای شرطهای پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتید مقادیر بولی می‌توانند `false` یا `true` باشند. فرض کنید که `var2` و `var3` دو مقدار بولی هستند.

عملگر	نام	مثال
&&	منطقی AND	var1 = var2 && var3
||	منطقی OR	var1 = var2 || var3
!	منطقی NOT	var1 =!var1

عملگر منطقی AND(&&)

اگر مقادیر دو طرف عملگر AND، `true` باشند عملگر AND مقدار `true` را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر یا هر دوی آنها `false` باشند مقدار `false` را بر می‌گرداند. در زیر جدول درستی عملگر AND نشان داده شده است:

X	Y	X && Y
true	true	true

true	false	false
false	true	false
false	false	false

برای درک بهتر تأثیر عملگر AND یاد آوری می‌کنم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عملوند مقدارشان true باشد. در غیر اینصورت نتیجه تمام ترکیبهای بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه‌ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
result = (age > 18) && (salary < 1000)
```

عملگر AND زمانی کارامد است که ما با محدوده خاصی از اعداد سرو کار داریم. مثلاً عبارت 10 <= x <= 100 بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100)
```

عملگر منطقی OR(||)

اگر یکی یا هر دو مقدار دو طرف عملگر OR، درست (true) باشد، عملگر OR مقدار true را بر می‌گرداند. جدول درستی عملگر OR در زیر نشان داده شده است:

X	Y	X || Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می‌کنید که عملگر OR در صورتی مقدار false را بر می‌گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100)
```

عملگر منطقی NOT(!)

برخلاف دو اپراتور OR و AND عملگر منطقی NOT، فقط به یک عملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

X	!X
true	false
false	true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگ‌تر یا مساوی ۱۸ نباشد.

```
isMinor = !(age >= 18)
```

عملگرهای بیتی

عملگرهای بیتی به شما اجازه می‌دهند که شکل باینری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم باینری و نحوه تبدیل اعداد دهدهی به باینری را از لینک زیر یاد بگیرید :

http://www.w3-farsi.com/?p=5698

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع Int به ۳۲ بیت یا ۴ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ وقتی به عنوان یک متغیر از نوع Int ذخیره می‌شود در کامپیوتر به صورت زیر خوانده می‌شود :

```
00000000000000000000000001100100
```

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی صفرهای سمت راست برای پر کردن بیتهایی است که عدد از نوع Int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست به چپ خوانده می‌شوند. عملگرهای بیتی Swift در جدول زیر نشان داده شده‌اند :

عملگر	نام	مثال
&	بیتی AND	x = y & z
|	بیتی OR	x = y | z
^	بیتی XOR	x = y ^ z
~	بیتی NOT	x = ~y

&=	بیتی-تخصیصی AND	x &= y
|=	بیتی-تخصیصی OR	x |= y
^=	بیتی-تخصیصی XOR	x ^= y

عملگر بیتی (&)AND

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیتها کار می‌کند. اگر مقادیر دو طرف آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عمگر بیتی AND در زیر آمده است:

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است :

```
var result = 5 & 3

print(result)
```

```
1
```

همانطور که در مثال بالا مشاهده می‌کنید نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد یک می‌شود. اجازه بدهید ببینیم که چطور این نتیجه را به دست می‌آید:

```
5:  00000000000000000000000000000101
3:  00000000000000000000000000000011
----------------------------------
1:  00000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح (Int) ۳۲ بیت است از صفر برای پر کردن بیتهای خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی (|)OR

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است :

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی OR در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید :

```
var result = 7 | 9

print(result)
```

```
15
```

وقتی که از عملگر بیتی OR برای دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه به دست آمده است؟

```
 7: 00000000000000000000000000000111
 9: 00000000000000000000000000001001
--------------------------------
15: 00000000000000000000000000001111
```

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح است.

عملگر بیتی XOR(^)

جدول درستی این عملگر در زیر آمده است :

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

در صورتیکه عملوندهای دو طرف این عملگر هر دو صفر یا هر دو یک باشند نتیجه صفر در غیر اینصورت نتیجه یک می‌شود. در مثال زیر تأثیر عملگر بیتی XOR را بر روی دو مقدار مشاهده می‌کنید :

```
var result = 5 ^ 7

print(result)
```

```
2
```

در زیر معادل باینری اعداد بالا (۵ و ۷) نشان داده شده است.

```
5: 00000000000000000000000000000101
7: 00000000000000000000000000000111
-----------------------------------
2: 00000000000000000000000000000010
```

با نگاه کردن به جدول درستی عملگر بیتی XOR، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر بیتی NOT(~)

این عملگر یک عملگر یگانی است و فقط به یک عملوند نیاز دارد. در زیر جدول درستی این عملگر آمده است:

NOT X	X
0	1
1	0

عملگر بیتی NOT مقادیر بیتها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است :

```
var result = ~7

print(result)
```

به نمایش باینری مثال بالا که در زیر نشان داده شده است توجه نمایید.

```
 7: 00000000000000000000000000000111
-----------------------------------
-8: 11111111111111111111111111111000
```

عملگر بیتی تغییر مکان (shift)

این نوع عملگرها به شما اجازه می‌دهند که بیتها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت ها را نشان می‌دهد.

عملگر	نام	مثال

>>	تغییر مکان به سمت چپ	x = y << 2
<<	تغییر مکان به سمت راست	x = y >> 2

عملگر تغییر مکان به سمت چپ

این عملگر بیتهای عملوند سمت چپ را به تعداد n مکان مشخص شده توسط عملوند سمت راست، به سمت چپ منتقل می‌کند. به عنوان مثال :

```
var result = 10 << 2

print(result)
```

```
40
```

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چپ منتقل کرده‌ایم، حال بیایید تأثیر این انتقال را بررسی کنیم :

```
10: 00000000000000000000000000001010
----------------------------------
40: 00000000000000000000000000101000
```

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان به سمت راست

این عملگر شبیه به عمگر تغییر مکان به سمت چپ است با این تفاوت که بیت‌ها را به سمت راست جا به جا می‌کند. به عنوان مثال :

```
var result = 100 >> 4

print(result)
```

```
6
```

با استفاده از عملگرتغییر مکان به سمت راست بیت‌های مقدار ۱۰۰ را به اندازه ۴ واحد به سمت چپ جا به جا می‌کنیم. اجازه بدهید تأثیر این جا به جایی را مورد بررسی قرار دهیم :

```
100: 00000000000000000000000001100100
----------------------------------
  6: 00000000000000000000000000000110
```

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

عملگرهای محدوده

Swift دارای سه نوع عملگر محدوده یا بازه می‌باشد که به کمک آن‌ها می‌توانید بازه‌ای از مقادیر را نشان دهید:

عملگر محدوده	توضیح	مثال
بسته	به صورت (a...b) تعریف می‌شود و مقادیر بین a و b را شامل می‌شود (مقادیر a و b را هم شامل می‌شود).	1...5 ، مقادیر ۱، ۲، ۳، ۴ و ۵ را شامل می‌شود.
نیمه باز	به صورت (a..<b) تعریف می‌شود و مقادیر از a تا b را شامل می‌شود (مقدار b جز بازه نیست).	1..<5، مقادیر ۱، ۲، ۳ و ۴ را شامل می‌شود.
یک طرف باز	به دو صورت زیر تعریف می‌شود: ...a : این حالت همه مقادیر از a تا آخر را شامل می‌شود a... : این حالت همه مقادیر از ابتدا تا a را شامل می‌شود.	...1، مقادیر ۱، ۲، ۳ و ... را شامل می‌شود. ...2، مقادیر ۱ و ۲ را شامل می‌شود.

از این نوع عملگرها بیشتر در دستورات شرطی استفاده می‌شود که در درس‌های آینده توضیح می‌دهیم.

عملگرهای متفرقه

سوییفت از عملگرهای دیگری نیز پشتیبانی می کند که در زیر می توانید آنها را مشاهده کنید:

عملگر	توضیح	مثال
عملگر منفی (-)	علامت یک متغیر عددی را می توان با استفاده از این عملگر تعیین کرد.	-۳ یا -۴
عملگر مثبت (+)	مقدار یک متغیر را بدون هیچ تغییری بر می گرداند.	+۴ معادل همان ۴ است.
عملگر سه تایی (?:)	اگر شرط (Condition) درست باشد، مقدار بعد از علامت ? در غیر اینصورت مقدار بعد از علامت : را بر می گرداند.	`Condition ? X : Y`

تقدم عملگرها

تقدم عملگرها، مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند، ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در Swift در محاسبات دارای حق تقدم هستند. به عنوان مثال :

```
var number :Int = 1 + 2 * 3 / 1
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه ۹ خواهد شد (۳=۲+۱ سپس ۹=۳×۳ و در آخر ۹=۹/۱). اما کامپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آنها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید. در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای Swift آمده است :

عملگر	تقدم
() [] . expr++ expr-	بالاترین
* & + - ! ~ ++expr -expr * / % + - >> << =< => < > == !=	
& ^ | && ||	
?:	
= += -= *= /= %= >>= <<= &=^= |=	
,	پایین ترین

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین‌ترین حق تقدم در محاسبات تأثیر می‌گذارند. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آنها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم :

```
var number :Int = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ))
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد توجه کنید. در این عبارت ابتدا مقدار داخلی‌ترین پرانتز مورد محاسبه قرار می‌گیرد یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال :

```
var number :Int = 3 * 2 + 8 / 4
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آنها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر number قرار می‌دهید.

گرفتن ورودی از کاربر

Swift از تابع readLine() برای گرفتن ورودی از کاربر استفاده می‌کند. این تابع، تمام کاراکترهایی را که شما در محیط کنسول تایپ می‌کنید تا زمانی که دکمه Enter را می‌زنید، می‌خواند. به برنامه زیر توجه کنید :

```
1  print("Enter your name: ", terminator:"")
2  var name = readLine()
3
4  print("Enter your age: ", terminator:"")
5  var age = readLine()
6
7  print("Enter your height: ", terminator:"")
8  var height = readLine()
9
10 //Print a blank line
11 print();
12
13 //Show the details you typed
14 print("Name is \(name!)")
15 print("Age is \(age!)")
16 print("Height is \(height!)")
```

با اجرای برنامه، نشان گر ماوس نمایش داده می‌شود و برنامه از شما می‌خواهد که اطلاعات را وارد کنید. برنامه از شما نام، سن و قدتان را می‌خواهد و شما باید بعد از وارد کردن اطلاعات هر بخش، دکمه Enter را بزنید. با زدن دکمه Enter آخر، خروجی به صورت زیر نمایش داده می‌شود:

```
Enter your name: John
Enter your age: 18
Enter your height: 160.5

Name is John
Age is 18
Height is 160.5
```

همانطور که در خطوط ۱۴، ۱۵ و ۱۶ کد بالا مشاهده می‌کنید، ما در جلوی نام هر متغیر یک علامت ! گذاشته‌ایم. دلیل این کار این است که تابع readLine() یک رشته اختیاری را بر می‌گرداند، در نتیجه شما برای باز کردن و نشان دادن مقدار این رشته باشد آن را unwrap یا

باز کنید و این کار هم با استفاده از علامت ! ممکن است. در همین حد کافیست که بدانید که تابع readLine() ورودی را از کاربر گرفته و آنها را به صورت رشته در متغیرهای age، name و height (خطوط ۲، ۵ و ۸) قرار می‌دهد. قبول دارید که سن و قد باید از نوع عددی باشند؟ اگر بخواهید فقط در محیط کنسول مقادیری را نمایش دهید که عددی یا غیر عددی بود مشکلی به وجود نمی‌آورد. مانند مثال بالا، که ما فقط از کاربر اطلاعاتی را گرفته و چاپ کرده‌ایم. ولی اگر بخواهیم دو عدد را با هم جمع کنیم، باید چکار کنیم. به کد زیر توجه کنید :

```
print("Enter Number1: ", terminator:"")
var number1 = readLine()

print("Enter Number2: ", terminator:"")
var number2 = readLine()

print()

print("Sum is : " , (number1! + number2!))
```

کد بالا را اجرا کرده و دو عدد را وارد نمایید:

```
Enter Number1: 10
Enter Number2: 5

Sum is :  105
```

همانطور که مشاهده می‌کنید بعد از وارد کردن اعداد ۱۰ و ۵ خروجی عدد ۱۰۵ می‌شود. البته در اصل عدد ۱۰۵ نیست، بلکه برنامه دو عدد را مانند دو رشته در نظر می‌گیرد و آنها را به هم می‌چسباند. اگر بخواهیم این دو عدد را واقعاً با هم جمع کنیم و عدد ۱۵ را به دست بیاوریم باید بعد از تابع readLine() از توابع تبدیل داده که در درس‌های قبل ذکر کردیم به صورت زیر استفاده نمایید:

```
print("Enter Number1: ", terminator:"")
let number1 = Int(readLine()!)

print("Enter Number2: ", terminator:"")
let number2 = Int(readLine()!)

print()

print("Sum is : ", (number1! + number2!) )
```

به خروجی کد بالا توجه کنید:

```
Enter Number1: 10
Enter Number2: 5

Sum is : 15
```

همانطور که گفتیم، تابع readLine() رشته دریافت می‌کند، برای تبدیل این رشته به عدد صحیح از تابع Int() در استفاده کرده‌ایم. در نتیجه تابع Int() رشته گرفته شده توسط تابع readLine() را به نوع صحیح تبدیل کرده و در متغیرهای number1 و number2 می‌گذارد. با این وجود می‌توانیم دو متغیر را که عددی هستند با هم جمع کنیم.

ساختارهای تصمیم

تقریبا همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. Swift راه های مختلفی برای رفع این نوع مشکلات ارائه می دهد. در این بخش با مطالب زیر آشنا خواهید شد :

- دستور if
- دستور if…else
- عملگر سه تایی
- دستور if چندگانه
- دستور if تو در تو
- عملگرهای منطقی
- دستور switch

دستور if

می‌توان با استفاده از دستور if و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور if ساده‌ترین دستور شرطی است که برنامه می‌گوید اگر شرطی برقرار است، کد معینی را انجام بده. ساختار دستور if به صورت زیر است :

```
if (condition)
{
   code to execute
}
```

قبل از اجرای دستور if ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدهید که نگاهی به نحوه استفاده از دستور if در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار number کمتر از ۱۰ و Goodbye World را اگر مقدار number از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```
//Declare a variable and set it a value less than 10
var number = 5

//If the value of number is less than 10
if (number < 10)
{
    print("Hello World.")
}
//Change the value of a number to a value which
// is greater than 10
number = 15

//If the value of number is greater than 10
if (number > 10)
{
    print("Goodbye World.")
```

```
}
```

```
Hello World.
Goodbye World.
```

در خط ۲ یک متغیر با نام number تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور if در خط ۵ می‌رسیم برنامه تشخیص می‌دهد که مقدار number از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد. بنابراین خط ۷ اجرا و پیغام Hello World چاپ می‌شود. حال مقدار number را به ۱۵ تغییر می‌دهیم (خط ۱۱). وقتی به دومین دستور if در خط ۱۴ می‌رسیم برنامه مقدار number را با ۱۰ مقایسه می‌کند و چون مقدار number یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۱۶). به این نکته توجه کنید که دستور if را می‌توان در یک خط نوشت :

```
if (number < 10) { print("Hello World.") }
```

شما می‌توانید چندین دستور را در داخل دستور if بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدنه دستور if هستند. نحوه تعریف چند دستور در داخل بدنه if به صورت زیر است :

```
if (condition)
{
    statement1
    statement2
    .
    .
    .
    statementN
}
```

این هم یک مثال ساده :

```
if (x > 10)
{
    print("x is greater than 10.")
    print("This is still part of the if statement.")
}
```

در مثال بالا اگر مقدار x از ۱۰ بزرگ‌تر باشد دو پیغام چاپ می‌شود. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطا شده و یافتن آن را سخت می‌کند.

دستور if…else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود. یعنی اگر حالتی برقرار بود، کار خاصی انجام شود. اما زمانی که شما بخواهید، اگر شرط خاصی برقرار شد، یک دستور و اگر برقرار نبود، دستور دیگر اجرا شود، باید از دستور if else استفاده کنید. ساختار دستور if else در زیر آمده است :

```
if (condition)
```

```
{
   code to execute if condition is true
}
else
{
   code to execute if condition is false
}
```

از کلمه کلیدی else نمی‌توان به تنهایی استفاده کرد بلکه حتماً باید با if به کار برده شود. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if…else آمده است.

```
1  var number = 5
2
3  //Test the condition
4  if (number < 10)
5  {
6      print("The number is less than 10.")
7  }
8  else
9  {
10     print("The number is either greater than or equal to 10.")
11 }
12
13 //Modify value of number
14 number = 15
15
16 //Repeat the test to yield a different result
17 if (number < 10)
18 {
19     print("The number is less than 10.")
20 }
21 else
22 {
23     print("The number is either greater than or equal to 10.")
24 }
```

```
The number is less than 10.
The number is either greater than or equal to 10.
```

در خط ۱ یک متغیر به نام number تعریف کرده‌ایم و در خط ۴ تست می‌کنیم که آیا مقدار متغیر number از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک if اجرا می‌شود (خط ۶) و اگر مقدار number را تغییر دهیم و به مقداری بزرگتر از ۱۰ تغییر دهیم (خط ۱۴)، شرط نادرست می‌شود (خط ۱۷) و کد داخل بلوک else اجرا می‌شود (خط ۲۳).

دستور if تو در تو

می‌توان از دستور if تو در تو در Swift استفاده کرد. یک دستور ساده if در داخل دستور if دیگر.

```
if (condition)
{
    code to execute

    if (condition)
    {
         code to execute
```

```
    }
    else if (condition)
    {
        if (condition)
        {
           code to execute
        }
    }
}
else
{
    if (condition)
    {
       code to execute
    }
}
```

اجازه بدهید که نحوه استفاده از دستور if تو در تو را نشان دهیم :

```
1  var age = 21
2
3  if (age > 12)
4  {
5      if (age < 20)
6      {
7          print("You are teenage")
8      }
9      else
10     {
11         print("You are already an adult.")
12     }
13 }
14 else
15 {
16     print("You are still too young.")
17 }
```

```
You are already an adult.
```

اجازه بدهید که برنامه را کالبد شکافی کنیم. ابتدا در خط ۱ یک متغیر به نام age تعریف می‌کنیم و مقدار آن را برابر ۲۱ قرار می‌دهیم. سپس به اولین دستور if می‌رسیم (خط ۳). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنه دستور if می‌شود در غیر اینصورت وارد بلوک else (خط ۱۴) مربوط به همین دستور if می‌شود.

حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنه اولین if شده‌اید. در بدنه اولین if یک دستور if دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد دستور You are teenage چاپ می‌شود (خط ۷) در غیر اینصورت دستور You are already an adult (خط ۱۱) و چون مقدار متغیر تعریف شده در خط 5 بزرگ‌تر از ۲۰ است پس دستور مربوط به بخش else خط ۱۱ چاپ می‌شود. حال فرض کنید که مقدار متغیر age کمتر از ۱۲ بود، در این صورت دستور بخش else خط ۱۶ یعنی You are still too young چاپ می‌شد. پیشنهاد می‌شود که از if تو در تو در برنامه کمتر استفاده کنید چون خوانایی برنامه را پایین می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید :

```
if (condition)
{
   code to execute
}
else
{
   if (condition)
   {
      code to execute
   }
   else
   {
      if (condition)
      {
         code to execute
      }
      else
      {
         code to execute
      }
   }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده‌تر کنید :

```
if (condition)
{
   code to execute
}
else if (condition)
{
   code to execute
}
else if (condition)
{
   code to execute
}
else
{
   code to execute
}
```

حال که نحوه استفده از دستور else if را یاد گرفتید باید بدانید که مانند else، else if نیز به دستور if وابسته است. دستور else if وقتی اجرا می‌شود که اولین دستور if اشتباه باشد حال اگر else if اشتباه باشد دستور else if بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور else اجرا می‌شود. برنامه زیر نحوه استفاده از دستور if else را نشان می‌دهد :

```
var choice = 2
```

```
if (choice == 1)
{
    print("You might like my black t-shirt.")
}
else if (choice == 2)
{
    print("You might be a clean and tidy person.")
}
else if (choice == 3)
{
    print("You might be sad today.")
}
else
{
    print("Sorry, your favorite color is not in the choices above.")
}
```

```
You might be a clean and tidy person
```

خروجی برنامه بالا به متغیر choice وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغامهای مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالتهای انتخاب نباشد کد مربوط به بلوک else اجرا می‌شود.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است :

عملگر	تلفظ	مثال	تأثیر
&&	and	`z = (x > 2) && (y < 10)`	مقدار Z در صورتی true است که هر دو شرط دو طرف عملگر مقدارشان true باشد. اگر فقط مقدار یکی از شروط false باشد مقدار z، false خواهد شد.
||	or	`z = (x > 2) || (y < 10)`	مقدار Z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان false باشد مقدار z، false خواهد شد.
!	not	`z = !(x > 2)`	مقدار Z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد.

به عنوان مثال جمله `z = (x > 2) && (y < 10)` را به این صورت بخوانید: "در صورتی مقدار z برابر true است که مقدار x بزرگتر از ۲ و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد.

عملگر منطقی || تأثیر متفاوتی نسبت به عملگر منطقی && دارد. نتیجه عملگر منطقی || برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی && و || را با هم ترکیب کرده و در یک عبارت به کار برد مانند :

```
if ( (x == 1) && ( (y > 3) || z < 10) )
{
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرطها استفاده می‌کنیم. در اینجا ابتدا عبارت `(y > 3) || (z < 10)` مورد بررسی قرار می‌گیرد. (به علت تقدم عملگرها) سپس نتیجه آن بوسیله عملگر && با نتیجه `(x == 1)` مقایسه می‌شود. حال بیایید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم :

```
1  print("Enter your age: ", terminator:"")
2  var age = Int(readLine()!)
3
4  print("Enter your gender (male/female):", terminator:"")
5  var gender = readLine()
6
7  if (age! > 12 && age! < 20)
8  {
9      if (gender == "male")
10     {
11         print("You are a teenage boy.")
12     }
13     else
14     {
15         print("You are not a teenage girl.")
16     }
17 }
18 else
19 {
20     print("You are not a teenager.")
21 }
```

```
Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی && را نشان می‌دهد (خط ۷). وقتی به دستور if می‌رسید (خط ۷) برنامه سن شما را چک می‌کند. اگر سن شما بزرگتر از ۱۲ و کوچکتر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو true باشد سپس کدهای داخل بلوک if اجرا می‌شوند. اگر نتیجه یکی از شروط false باشد کدهای داخل بلوک else اجرا می‌شود.

عملگر && عملوند سمت چپ را مورد بررسی قرار می‌دهد. اگر مقدار آن false باشد دیگر عملوند سمت راست را بررسی نمی‌کند و مقدار false را بر می‌گرداند. بر عکس عملگر || عملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن true باشد سپس عملوند سمت راست را نادیده می‌گیرد و مقدار true را بر می‌گرداند.

```
if ((x == 2) && (y == 3))
{
    //Some code here
}

if ((x == 2)  || (y == 3))
{
    //Some code here
}
```

شما می‌توانید از عملگرهای && و || بیتی هم در شرط ها استفاده کنید. این عملگرها دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ false باشد عملوند سمت چپ به وسیله عملگر بیتی & ارزیابی می‌شود. اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی && و || به جای عملگرهای بیتی && و || بهتر خواهد بود:

```
if ((age > 12) && (age < 20))
```

یکی دیگر از عملگرهای منطقی عملگر ! است که نتیجه یک عبارت را خنثی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (x != 2)
{
    print("x is not equal to 2.")
}
```

اگر نتیجه عبارت x == 2 برابر false باشد عملگر ! آن را true می‌کند. می توان خط ۷ کد بالا را به صورت زیر هم نوشت:

```
if(12...20 ~= age!)
{
    // Some Code
}
```

دستور Switch

در Swift ساختاری به نام switch وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور switch معادل دستور if تو در تو است با این تفاوت که در دستور switch متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغییر نیستند. در زیر نحوه استفاده از دستور switch آمده است :

```
switch (testVar)
{
   case compareVal1:
       code to execute if testVar == compareVa11
   case compareVa12:
       code to execute if testVar == compareVa12
   .
   .
```

```
    .
    case compareVa1N:
        code to execute if testVer == compareVa1N
    default:
        code to execute if none of the values above match the testVar
}
```

ابتدا یک مقدار در متغیر switch که در مثال بالا testVar است قرار می‌دهید. این مقدار با هر یک از عبارتهای case داخل بلوک switch مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات case برابر بود کد مربوط به آن case اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور case از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. دستور switch یک بخش default دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات case برابر نباشد. دستور default اختیاری است و اگر از بدنه switch حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور switch توجه کنید :

```
print("What's your favorite pet?")
print("[1] Dog")
print("[2] Cat")
print("[3] Rabbit")
print("[4] Turtle")
print("[5] Fish")
print("[6] Not in the choices")
print("\nEnter your choice: ", terminator:"")

var choice = Int(readLine()!)

switch (choice!)
{
    case 1:
        print("Your favorite pet is Dog.")
    case 2:
        print("Your favorite pet is Cat.")
    case 3:
        print("Your favorite pet is Rabbit.")
    case 4:
        print("Your favorite pet is Turtle.")
    case 5:
        print("Your favorite pet is Fish.")
    case 6:
        print("Your favorite pet is not in the choices.")
    default:
        print("You don't have a favorite pet.")
}
```

```
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
```

```
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور switch با مقادیر case مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر case ها برابر نبود دستور default اجرا می‌شود. یکی دیگر از ویژگیهای دستور switch این است که شما می‌توانید از دو یا چند مقدار برای مقایسه استفاده کنید. در مثال زیر اگر مقدار number عدد ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که مقادیر با استفاده از علامت , از هم جدا می‌شوند:

```
switch(number)
{
    case 1, 2, 3:
        print("This code is shared by three values.")
}
```

همانطور که قبلاً ذکر شد دستور switch معادل دستور if else است. برنامه بالا را به صورت زیر نیز می‌توان نوشت :

```
if (choice! == 1)
{
   print("Your favorite pet is Dog.")
}
else if (choice! == 2)
{
   print("Your favorite pet is Cat.")
}
else if (choice! == 3)
{
   print("Your favorite pet is Rabbit.")
}
else if (choice! == 4)
{
   print("Your favorite pet is Turtle.")
}
else if (choice! == 5)
{
   print("Your favorite pet is Fish.")
}
else if (choice! == 6)
{
   print("Your favorite pet is not in the choice!s.")
}
else
{
   print("You don't have a favorite pet.")
}
```

کد بالا دقیقاً نتیجه‌ای مانند دستور switch دارد. دستور default معادل دستور else می‌باشد. گاهی اوقات لازم است که با وجود اجرا شدن یک case دوست دارید، بقیه case ها هم اجرا شوند. در این صورت می‌توانید از عبارت fallthrough استفاده کنید :

```
var number :Int = 10

switch(number)
{
    case 11, 10, 20:
        print("first case executed!")
    case 45:
        print("second case executed!")
    default:
        print("default statement")
}
```

```
first case executed!
```

در کد بالا از دستور fallthrough استفاده نشده، در نتیجه برنامه بعد از اجرای دستور case ی که صحیح است، از دستور Switch خارج می‌شود. اما در کد زیر از دستور fallthrough استفاده شده است و در نتیجه دستورات مربوط به بقیه case ها هم اجرا می‌شوند:

```
var number :Int = 10

switch(number)
{
    case 11, 10, 20:
        print("first case executed!")
        fallthrough
    case 45:
        print("second case executed!")
    default:
        print("default statement")
}
```

```
first case executed!
second case executed!
```

عملگر شرطی

عملگر شرطی (:?) در Swift مانند دستور شرطی if…else عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی Swift است که نیاز به سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدهید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
1  var pet1  :String = "puppy"
2  var pet2  :String = "kitten"
3  var type1 :String
4  var type2 :String
5
6  type1 = (pet1 == "puppy" ) ? "dog" : "cat"
7  type2 = (pet2 == "kitten") ? "cat" : "dog"
```

```
 8
 9  print(type1)
10  print(type2)
```

```
dog
cat
```

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می‌دهد. خط ۶ به صورت زیر ترجمه می‌شود: اگر مقدار pet1 برابر با puppy سپس مقدار dog را در type1 قرار بده در غیر این صورت مقدار cat را type1 قرار بده. خط ۷ به صورت زیر ترجمه می‌شود: اگر مقدار pet2 برابر با kitten سپس مقدار cat را در type2 قرار بده در غیر این صورت مقدار dog. حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```
if (pet1 == "puppy")
{
   type1 = "dog"
}
else
{
   type1 = "cat"
}
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می‌آورد.

تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید مانند مثال زیر :

```
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
print("Hello World.")
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای نوشتن کدهای بالا استفاده از حلقه‌ها است. حلقه‌ها در Swift عبارتند از :

- for-in
- do while
- repeat…while

حلقه While

ابتدایی‌ترین ساختار تکرار در Swift حلقه While است. ابتدا یک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای درون بلوک اجرا می‌شوند. ساختار این حلقه به صورت زیر است :

```
while(condition)
{
    // code to loop
}
```

می‌بینید که ساختار While مانند ساختار if بسیار ساده است. ابتدا یک شرط را که نتیجه آن یک مقدار بولی است، می‌نویسیم. اگر نتیجه درست یا true باشد، سپس کدهای داخل بلوک While اجرا می‌شوند. اگر شرط غلط یا false باشد وقتی که برنامه به حلقه While برسد هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقه باید مقادیر داخل حلقه While اصلاح شوند.

به یک متغیر شمارنده در داخل بدنه حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدنه باید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه While آمده است :

```
1  var counter = 1
2
3  while (counter <= 10)
4  {
5      print("Hello World!")
6      counter = counter+1
7  }
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

برنامه بالا ۱۰ بار پیغام Hello World! را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم، مجبور بودیم تمام ۱۰ خط را تایپ کنیم. اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۱ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد، نمی‌توان در شرط از آن استفاده کرد.

در خط ۳ حلقه While را وارد می‌کنیم. در این حلقه ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۶). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بینهایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت < از => استفاده شده است. اگر از علامت < استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ برسد false می‌شود چون 10 < 10 نیست. اگر می‌خواهید یک حلقه بی نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
   //code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

repeat…while

حلقه repeat while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه repeat while به صورت زیر است :

```
repeat
{
   //code to repeat

} while (condition)
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدنه حداقل یکبار اجرا می‌شوند. برخلاف حلقه while که اگر شرط نادرست باشد دستورات داخل بدنه اجرا نمی‌شوند. یکی از موارد برتری استفاده از حلقه repeat while نسبت به حلقه while، زمانی است که، شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید :

استفاده از while

```
//while version

print("Enter a number greater than 10: ", terminator:"")
var number = Int(readLine()!)

while(number! < 10)
{
    print("Enter a number greater than 10: ", terminator:"")
    number = Int(readLine()!)
}
```

استفاده از repeat while

```
//do while version

var number :Int?
repeat
{
    print("Enter a number greater than 10: ", terminator:"")
    number = Int(readLine()!)
}
```

```
while(number! < 10)
```

مشاهده می‌کنید که از کدهای کمتری در بدنه repeat while نسبت به while استفاده شده است.

حلقه for…in

یکی دیگر از ساختارهای تکرار، حلقه for است. این حلقه عملی شبیه به حلقه while انجام می‌دهد و از آن برای گردش در میان یک مجموعه (collection) یا محدوده (range) استفاده می شود. ساختار حلقه for به صورت زیر است :

```
for (item in collection/range)
{
    //Some Code
}
```

در زیر یک مثال از حلقه for آمده است:

```
1  for i in 1...10
2  {
3      print("Number \(i)")
4  }
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه for چاپ می کند. در خط ۱، همانطور که مشاهده می کنید عبارت ۱...۱۰ آمده است که در اصل یک محدود است. برای ایجاد یک محدوده در Swift ابتدا و انتهای مجموعه را مشخص کرده و سپس با استفاده از علامت ... از هم جدا می کنیم. عبارت ۱...۱۰ به معنای محدود اعداد از ۱ تا ۱۰ است. در کل معنای خط ۱ این است که به ازای اعداد ۱ تا ۱۰ ... فلان کار را انجام بده. چه کاری؟ کاری که در خط ۳ خواسته ایم و آن چاپ مقدار اعداد این محدود است. در اصل در خط ۱ با هر تکرار یکی از اعداد مجموعه در داخل متغیر موقتی i قرار می گیرد و در خط ۳ چاپ می شود و این کار تا چاپ آخرین مقدار یعنی ۱۰ ادامه می یابد. اگر بخواهید عدد انتهای محدود یعنی ۱۰ چاپ نشود باید از کلمه < استفاده نمایید:

```
for i in 1..<10
```

از حلقه for برای چاپ حروف یک رشته هم می توان استفاده کرد. چون رشته یک مجموعه از کاراکترهاست:

```
for letter in "Hello World!"
{
    print(letter)
}
```

```
H
```

```
e
l
l
o

W
o
r
l
d
!
```

خارج شدن از حلقه با استفاده از break و continue

گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت.

برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد :

```swift
print("Demonstrating the use of break\n")

for x in 1...9
{
    if (x == 5)
    {
        break
    }
    print("Number \(x)")
}

print("\nDemonstrating the use of continue\n")

for x in 1...9
{
    if (x == 5)
    {
        continue
    }
    print("Number \(x)")
}
```

```
Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7
Number 8
```

```
Number 9
```

در این برنامه از حلقه for برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای for از حلقه‌های while و repeat…while استفاده می‌شد نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۵) آمده است وقتی که مقدار x به عدد ۵ رسید سپس دستور break اجرا شود (خط ۷).

حلقه بلافاصله متوقف می‌شود حتی اگر شرط 10 > x برقرار باشد. از طرف دیگر در خط ۱۶ حلقه for فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد (وقتی مقدار x برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند).

آرایه ها

آرایه نوعی متغیر است که لیستی از آدرسهای مجموعه‌ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آنها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان همه آنها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است :

```
var arrayName = [datatype]()
```

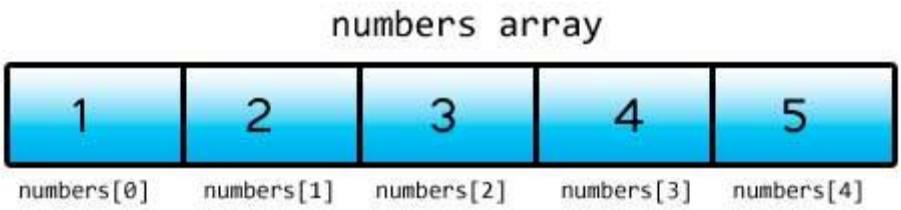
arrayName که نام آرایه را نشان می‌دهد. Datatype نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. کروشه که نوع داده در داخل آن قرار می‌گیرد و نشان دهنده استفاده از آرایه است و در نهایت علامت‌های باز و بسته پرانتز که در ادامه در مورد آنها توضیح می‌دهیم. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌ای که اعداد را در خود ذخیره می‌کند. برای مثال، فرض کنید می‌خواهیم یک آرایه تعریف کنیم که مقادیر صحیح را در خود ذخیره کند:

```
var numbers = [Int] ()
```

در کد بالا ما یک آرایه خالی تعریف کرده‌ایم که قرار است اعداد صحیح را در خود ذخیره کند. حال چطور مقادیرمان را به این آرایه اضافه کنیم؟ برای این منظور از عملگر =+ استفاده می‌کنیم:

```
numbers += [1]
numbers += [2]
numbers += [3]
numbers += [4]
numbers += [5]
```

همانطور که در کد بالا مشاهده می‌کنید بعد از عملگر =+، مقادیری که قرار است به آرایه اضافه شوند را، در داخل کروشه می‌نویسیم. توجه کنید که هر عنصر از آرایه دارای یک اندیس است. اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است پس ۱-۵ برابر است با ۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید :



به هر یک از اجزاء آرایه و اندیسهای داخل کروشه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیسها را از ۱ شروع کنند. اگر بخواهید به یکی از اجزائ آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیسهای آرایه شما نباشد با پیغام خطای Index out of range مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید که وجود ندارد. یکی دیگر از راه‌های تعریف سریع و مقدار دهی یک آرایه به صورت زیر است :

```
var arrayName: [datatype] = [val1, val2, ..., valN]
```

در این روش شما می‌توانید فوراً بعد از تعریف اندازه آرایه مقادیر را در داخل کروشه قرار دهید. به یاد داشته باشید که هر کدام از مقادیر را با استفاده از کاما از هم جدا کنید. به مثال زیر توجه کنید :

```
var numbers: [Int] = [ 1, 2, 3, 4, 5 ]
```

این مثال با مثال قبل هیچ تفاوتی ندارد و تعداد خطهای کدنویسی را کاهش می‌دهد. شما می‌توانید با استفاده از اندیس به مقدار هر یک از اجزاء آرایه دسترسی یابید و آنها را به دلخواه تغییر دهید. یکی دیگر از راه‌های تعریف آرایه در زیر آمده است. شما می‌توانید هر تعداد عنصر را که خواستید در آرایه قرار دهید بدون اینکه نوع آرایه را مشخص کنید. به عنوان مثال :

```
var numbers = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

در این مثال ما ۱۰ مقدار را به آرایه اختصاص داده‌ایم. روش دیگر برای تعریف آرایه به صورت زیر است:

```
var arrayname :[dataType] = Array(repeating: ... , count: ...)
```

در این روش بعد از علامت مساوی کلمه کلیدی Array را نوشته و در داخل پرانتزها از دو کلمه repeating و count استفاده می‌کنید. کلمه count، تعداد عناصر آرایه را مشخص می‌کند و کلمه repeating هم یک مقدار اولیه را در داخل عناصر آرایه قرار می‌دهد. به کد زیر توجه کنید:

```
var numbers :[Int] = Array(repeating: 0, count: 5)
```

کد بالا یک آرایه با نام number ایجاد کرده که دارای ۵ خانه است و در داخل هر خانه هم عدد ۰ قرار دارد:

```
[0, 0, 0, 0, 0]
```

حال چگونه به همچین آرایه‌ای، مقادیری را که می‌خواهیم، اختصاص دهیم؟ به کد زیر توجه کنید:

```
1  var numbers :[Int] = Array(repeating: 0, count: 5)
2
3  print("Element 4 before initialize : ", numbers[3])
```

```
4
5  numbers[0] = 1
6  numbers[1] = 2
7  numbers[2] = 3
8  numbers[3] = 4
9  numbers[4] = 5
10
11 print("Element 4 after initialize : ", numbers[3])
```

در خط ۱ کد بالا یک آرایه تعریف و تمام ۵ خانه آن را با عدد صفر مقداردهی کرده‌ایم. برای اطمینان در خط ۳ مقدار چهارمین عنصر را چاپ کرده‌ایم. همانطور که در خروجی مشاهده می‌کنید، عدد ۰ نمایش داده می‌شود. در ادامه و در خطوط ۹-۵، آرایه را با مقادیر جدید پر می‌کنیم. در این خطوط ابتدا نام آرایه و در داخل کروشه اندیس خانه‌ای که قرار است مقداری را در داخل آن قرار دهیم، می‌نویسیم. سپس با استفاده از علامت = مقدار را به آن خانه اختصاص می‌دهیم. خانه چهارم آرایه که اندیس آن ۳ است را با مقدار ۴، مقداردهی و در خط ۱۱ بار دیگر مقدار این خانه را چاپ می‌کنیم. مشاهده می‌کنید که در خروجی، عدد ۴ نمایش داده می‌شود. یعنی شکل نهایی این آرایه به صورت زیر است:

```
1, 2, 3, 4, 5
```

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آنها حساب می‌شود:

```
1  var numbers :[Int?] = Array(repeating: 0, count: 5)
2  var total   = 0
3  var average :Double
4
5  for i in 0..<numbers.count
6  {
7      print("Enter a number: ", terminator:"")
8      numbers[i] = Int(readLine()!)
9  }
10 for i in 0..<numbers.count
11 {
12     total += numbers[i]!
13 }
14
15 average = Double(total / numbers.count)
16
17 print("Average = \(average)", terminator:"")
```

```
Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط ۱ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. هنگام تعریف این آرایه از علامت ? استفاده کرده‌ایم و با این کار به برنامه می گوییم که ممکن است در ادامه یا در هنگام اجرای برنامه مقادیری را به خانه‌های آرایه که همه با ۰ پر شده‌اند، اختصاص دهیم. خطوط ۲ و ۳ متغیرهایی تعریف شده‌اند که از آنها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه

total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۵ تا ۹ یک حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. کلاً کار این حلقه این است که مقادیر را از کاربر می‌گیرد و در داخل خانه‌های آرایه قرار می‌دهد. از خاصیت count آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه for مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت count آرایه کار راحت‌تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می‌شود. در خط ۸ ورودی دریافت شده از کاربر به نوع Int تبدیل و در آرایه ذخیره می‌شود. اندیس استفاده شده در number (خط ۸) مقدار i جاری در حلقه است. برای مثال در ابتدای حلقه مقدار i صفر است بنابراین وقتی در خط ۸ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر صفر می‌شود. در تکرار بعدی i یک واحد اضافه می‌شود و در نتیجه در خط ۸ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌یابد. در خطوط ۱۳-۱۰ از حلقه for دیگر برای دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر total اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۱۵). مقدار total را بر تعداد اجزای آرایه (تعداد عددها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت count آرایه استفاده کرد. توجه کنید که در خط ۱۵ ما حاصل عبارت را به نوع Double تبدیل کرده‌ایم بنابراین نتیجه عبارت یک مقدار از نوع double خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع Double تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۱۷ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغیر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاسها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آنها استفاده کنید بسیار مهم است.

آرایه های چند بعدی

آرایه‌های چند بعدی، آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آنها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدادی ستون و سطر تصور کنید. با افزایش اندیسها اندازه ابعاد آرایه نیز افزایش می‌یابد و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است :

```
var arrayName = [[typeArray]]()
```

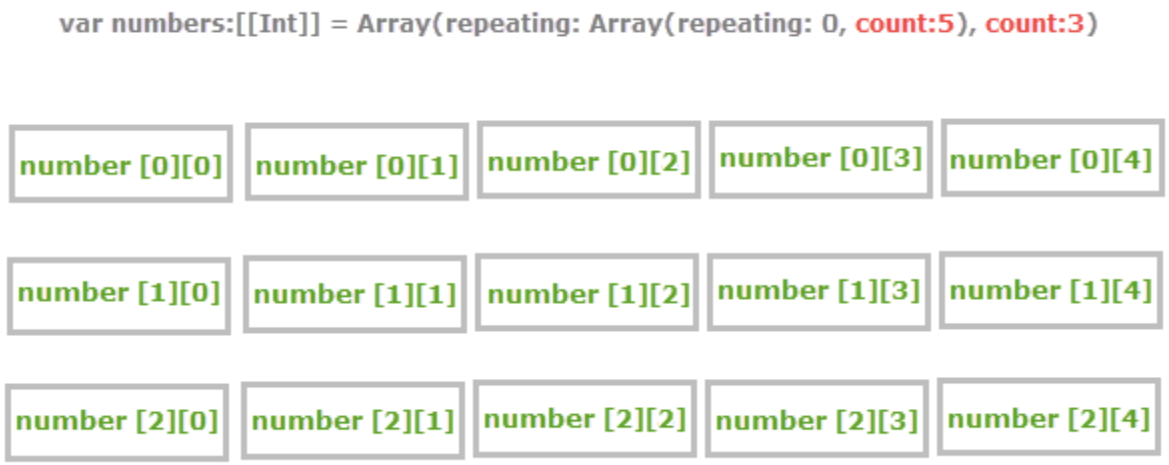
همانطور که قبلا گفتیم، وجود کروشه به معنای آرایه است. در کد بالا، [[typeArray]] به معنای آرایه ای از آرایه ها است. کد بالا را به صورت زیر هم می‌توان نوشت:

```
var arrayName :[[typeArray]] = [[array1], [array2]]
```

روش دیگر برای ایجاد آرایه دو بعدی به صورت زیر است:

```
var arrayName :[[typeArray]] = Array(repeating: Array(repeating: value that fill array,
                                                      count:column), count:row)
```

می‌توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی اندیس سطر (row) و دیگری اندیس ستونی (column) است که آن عنص درآن قرار دارد. البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به row، چون ۳ سطر یا آرایه داریم و مقدار ۵ را به column چون هر آرایه ۵ ستون دارد، اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ می خواهیم به سه روش بالا یک آرایه تعریف کنیم که دارای ۴ ستون و 2 سطر باشد. به روش اول توجه کنید:

```
1  var numbers = [[Int]]()
2
3  var row1 = [Int]()
4  row1 += [1]
5  row1 += [2]
6  row1 += [3]
7  row1 += [4]
8
9  var row2 = [Int]()
10 row2 += [5]
11 row2 += [6]
12 row2 += [7]
13 row2 += [8]
14
15 numbers += [row1]
16 numbers += [row2]
```

در روش بالا و خط ۱، ما یک آرایه دو بعدی تعریف کرده ایم. سپس در خطوط 3-7 و ۹-۱۲ دو آرایه به نام های row1 و row2 ایجاد و آنها را در خطوط ۱۵ و 16 به آرایه دو بعدی مان اضافه نموده ایم.

```
var numbers :[[Int]] = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

روش سوم

```
var numbers:[[Int]] = Array(repeating: Array(repeating: 0, count:4), count:2)

numbers[0][0] = 1
numbers[0][1] = 2
numbers[0][2] = 3
numbers[0][3] = 4

numbers[1][0] = 5
numbers[1][1] = 6
numbers[1][2] = 7
numbers[1][3] = 8
```

در روش سوم که از کلاس Array استفاده کرده‌ایم باید مقدار دهی به عناصر را به صورت دستی انجام داد. همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیسهای row و column و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه for تو در تو است:

```
var numbers :[[Int]] = [
                            [ 1,  2,  3,  4, 5 ],
                            [ 6,  7,  8,  9, 10],
                            [11, 12, 13, 14, 15]
                        ]

for row in 0..<numbers.count
{
    for column in 0..<numbers[row].count
    {
        print(numbers[row][column], terminator:" ")
    }
    print()
}
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که در مثال بالا نشان داده شده است با استفاده از یک حلقه ساده for نمی‌توان به مقادیر دسترسی یافت بلکه به یک حلقه for تو در تو نیاز داریم. به زبان ساده در کد بالا دو حلقه for داریم. یکی برای به دست آوردن تعداد آرایه‌ها و دیگری برای به دست آوردن ستون یا تعداد عناصر آرایه‌ها. در اولین حلقه for، با استفاده از خاصیت count، اندیس آرایه‌های موجود در آرایه numbers که در اصل همان تعداد آرایه‌ها می‌باشد را به دست می‌آوریم. این خط را می‌توان به صورت زیر هم نوشت :

```
for(row in 0...2)
```

چرا ۰..۲؟ چون اندیس آرایه‌ها هم از صفر شروع می‌شود. یعنی اندیس اولین آرایه دارای اندیس ۰ و ... . در حلقه for دوم با استفاده از خاصیت count، تعداد ستون‌های یک آرایه را به دست می‌آوریم. و چون تعداد عناصر یا ستون‌های همه آرایه‌ها برابر ۵ است پس عبارت numbers[row].count عدد ۵ را بر می‌گرداند. در نتیجه خط ۱۱ را می‌توان به صورت زیر هم نوشت :

```
for column in 0..<5
```

در مجموع می‌توان گفت که row اعداد ۰، ۱ و ۲ را در خود جای می‌دهد و column هم اعداد ۰، ۱، ۲، ۳ و ۴. در نتیجه وقتی که مقدار ردیف 0 (row) باشد، حلقه دوم از [۰][۰] تا [۴][۰] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان می‌دهیم، اگر مقدار ردیف (row) برابر ۰ و مقدار ستون (column) برابر ۰ باشد، مقدار عنصری که در ستون ۱ و ردیف ۱ (numbers[0][0]) قرار دارد نشان داده خواهد شد که در مثال بالا، عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور ()print که به برنامه اطلاع می‌دهد که به خط بعد برود (خط ۱۷). سپس حلقه با اضافه کردن یک واحد به مقدار row این فرایند را دوباره تکرار می‌کند. سپس دومین حلقه for اجرا شده و مقادیر دومین ردیف نمایش داده می‌شود. حال بیایید آنچه را از قبل یاد گرفته‌ایم در یک برنامه به کار بریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما می‌گیرد و معدل سه دانش آموز را حساب می‌کند.

```
1  var studentGrades :[[Double?]] = Array(repeating: Array(repeating: 0, count:4),
2  count:3)
3  var total: Double
4
5  for student in 0..<studentGrades.count
6  {
7      total = 0.0
8
9      print("Enter grades for Student \(student+1)")
10
11     for grade in 0..<studentGrades[student].count
12     {
13         print("Enter Grade #\(grade): ", terminator:"")
14         studentGrades[student][grade] = Double(readLine()!)
15         total += studentGrades[student][grade]!
16     }
17
18     print("Average is ", total / Double(studentGrades[student].count))
19     print()
   }
```

```
Enter grades for Student 1
Enter Grade #1: 92
Enter Grade #2: 87
Enter Grade #3: 89
Enter Grade #4: 95
Average is 90.75

Enter grades for Student 2
Enter Grade #1: 85
Enter Grade #2: 85
Enter Grade #3: 86
Enter Grade #4: 87
Average is 85.75

Enter grades for Student 3
Enter Grade #1: 90
Enter Grade #2: 90
Enter Grade #3: 90
Enter Grade #4: 90
```

```
Average is 90.00
```

در برنامه بالا یک آرایه دو بعدی از نوع Double تعریف شده است (خط ۱). این آرایه ۳ آرایه دارد و هر آرایه ۴ عنصر. در کل ۳ دانش آموز و هر دانش آموز ۴ درس. همچنین یک متغیر به نام total تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم (خط ۲). حال وارد حلقه for تو در تو می‌شویم (خط ۴). در اولین حلقه for تعداد دانش آموزان را با استفاده از خاصیت count به دست می‌آوریم. در نتیجه متغیر student اعداد ۰، ۱ و ۲ را در خود ذخیره می‌کند. وارد بدنه حلقه for می‌شویم. در خط ۶ مقدار متغیر total را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید .(student + 1) عدد ۱ را به student اضافه کرده‌ایم تا به جای نمایش student 0، با student 1 شروع شود، تا طبیعی‌تر به نظر برسد.

سپس به دومین حلقه for در خط ۱۰ می‌رسیم. در این حلقه یک متغیر شمارنده به نام grade تعریف می‌کنیم و تعداد عناصر آرایه را با استفاده از studentGrades[student].count را در آن قرار می‌دهیم. این متغیر اعداد ۰، ۱، ۲ و ۳ را در خود جای می‌دهد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر total اضافه می‌شود. وقتی همه نمره‌ها وارد شدند، متغیر total هم جمع همه نمرات را نشان می‌دهد. در خط ۱۷ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن total (جمع) بر تعداد نمرات به دست می‌آید.

تابع

توابع به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. توابع دارای آرگومانهایی هستند که وظیفه تابع را مشخص می‌کنند. می‌توان یک تابع را در داخل تابع دیگر تعریف کرد. وقتی که شما در برنامه یک تابع را صدا می‌زنید برنامه به قسمت تعریف تابع رفته و کدهای آن را اجرا می‌کند.

پارامترها همان چیزهایی هستند که تابع منتظر دریافت آنها است.

آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک تابع به صورت زیر است :

```
func functionName()
{
   //code to execute
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک تابع برای چاپ یک پیغام در صفحه نمایش استفاده شده است :

```
1  func printMessage()
2  {
3      print("Hello World!")
4  }
5
6  printMessage()
```

```
Hello World!
```

در خطوط ۴-۱ یک تابع تعریف کرده‌ایم. مکان تعریف آن در برنامه مهم نیست. کار این تابع چاپ یک پیغام تعریف است. در تعریف تابع بالا، ابتدا کلمه کلیدی func آمده است که نشان دهنده آن است که ما می‌خواهیم یک تابع تعریف کنیم. نام تابع ما printMessage() است.

به این نکته توجه کنید که در نامگذاری تابع از روش کوهان شتری (اولین کلمه با حروف کوچک نوشته می‌شود و بقیه کلمات با حرف بزرگ شروع می‌شوند) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص توابع استفاده کنید. بهتر است در نامگذاری توابع از کلماتی استفاده شود که کار آن تابع را مشخص می‌کند مثلاً نام‌هایی مانند goToBed یا openDoor.

همچنین به عنوان مثال اگر مقدار برگشتی تابع یک مقدار بولی باشد می‌توانید اسم تابع خود را به صورت یک کلمه سوالی انتخاب کنید مانند isLeapyear یا ... isTeenager ولی از گذاشتن علامت سؤال در آخر اسم تابع، خودداری کنید. دو پرانتزی که بعد از نام می‌آید، نشان دهنده آن است که، نام متعلق به یک تابع است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درسهای آینده در مورد توابع بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه تابع را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در خط ۶، تابعی که ایجاد کرده‌ایم را، صدا می‌زنیم. برای صدا زدن یک تابع کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر تابع دارای پارامتر باشد باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می‌دهیم. با صدا زدن یک تابع کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای تابع printMessage() برنامه از خط ۶ به محل تعریف تابع printMessage() می‌رود. مثلاً وقتی ما تابع printMessage() را در خط ۶ صدا می‌زنیم برنامه از خط ۶ به خط ۱، یعنی جایی که تابع تعریف شده می‌رود و کدهای بدنه آن یعنی خط ۳ را اجرا می‌کند.

مقدار برگشتی از یک تابع

توابع، می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک تابع است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی تابع است. نکته مهم در مورد یک تابع، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک تابع آسان است. کافیست در تعریف تابع به روش زیر عمل کنید :

```
fun methodName() -> returntype
{
   return value
}
```

returnType در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (Int، Float و ...). در داخل بدنه تابع کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن، یک مقدار است را می‌نویسیم. نوع این مقدار برگشتی می تواند از انواع ساده و یا اختیاری بوده و در هنگام نامگذاری تابع و بعد از پرانتزها و علامت -> ذکر شود. مثال زیر یک تابع که دارای مقدار برگشتی است را نشان می‌دهد:

```
1  func calculateSum() -> Int
2  {
3      let firstNumber  :Int = 10
4      let secondNumber :Int = 5
5
6      return firstNumber + secondNumber
7  }
8
9  var result = calculateSum()
10
11 print("Sum is \(result).")
```

```
Sum is 15.
```

همانطور که در خط ۱ مثال فوق مشاهده می‌کنید، هنگام تعریف تابع و بعد از علامت پرانتز بسته، از Int -> استفاده کرده‌ایم که نشان دهنده آن است که تابع ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۳ و ۴ دو ثابت تعریف و مقدار دهی شده‌اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها فقط در تابعی که در آن تعریف شده‌اند، قابل استفاده هستند. در خط ۶ جمع دو ثابت، توسط دستور return برگشت داده می‌شود. یک متغیر به نام result در خط ۹ تعریف می‌کنیم و تابع ()calculateSum را فراخوانی می‌کنیم.

تابع ()calculateSum مقدار ۱۵ را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط ۱۱ مقدار ذخیره شده در متغیر result چاپ می‌شود. تابعی که در این مثال ذکر شد، تابع کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در تابع بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این تابع در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک تابع از دستور if یا switch استفاده کنیم، باید خود این دستورات را با کلمه return برگشت دهیم. برای درک بهتر این مطلب به مثال زیر توجه کنید :

```
1  func getNumber() -> Int
2  {
3      print("Enter a number greater than 10: ", terminator:"")
4      let number = Int(readLine()!)
5
6      if (number! > 10)
7      {
8          return number!
9      }
10     else
11     {
12         return 0
13     }
14 }
15
16 var result = getNumber()
```

```
17
18  print("Result = \(result).")
```

```
Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط ۱۴-۱ یک تابع با نام getNumber() تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد تابع مقدار صفر را بر می‌گرداند. اگر قسمت else دستور if و یا دستور return را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطا مواجه می‌شویم.

چون اگر شرط دستور if نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت else می‌رود تا مقدار ۰ را بر گرداند و چون قسمت else حذف شده است برنامه با خطا مواجه می‌شود و همچنین اگر دستور return حذف شود چون برنامه نیاز به مقدار برگشتی دارد، پیغام خطا می‌دهد.

پارامتر و آرگومان

پارامترها، داده‌های خامی هستند که، تابع آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید، در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که، شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک تابع می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک تابع با N پارامتر نشان داده شده است :

```
fun functionName(param1: datatype , param2: datatype , ... paramN: datatype)
{
   // code to execute
}
```

پارامترها، بعد از نام تابع و بین پرانتزها قرار می‌گیرند. بر اساس کاری که تابع انجام می‌دهد، می‌توان تعداد پارامترهای زیادی به تابع اضافه کرد. بعد از فراخوانی یک تابع باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها، مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطا می‌شود. اجازه بدهید که یک مثال بزنیم :

```
1   func calculateSum(number1: Int, number2: Int) -> Int
2   {
3       return number1 + number2
4   }
5
6   print("Enter the first number: ", terminator:"")
7   var num1 = Int(readLine()!)
8   print("Enter the second number: ", terminator:"")
9   var num2 = Int(readLine()!)
10
11  print("Sum = ", calculateSum(number1: num1! , number2: num2!))
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک تابع به نام calculateSum() (خطوط ۴-۱) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این تابع مقدار دو عدد صحیح را با هم جمع می‌کند، پس نوع برگشتی ما نیز باید Int باشد. تابع دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (Int) دریافت می‌کنند. در بدنه تابع دستور return نتیجه جمع دو عدد را بر می‌گرداند. در داخل خطوط ۹-۶ برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال تابع را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به تابع تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه تابع ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی تابع دقیقاً با ترتیب قرار گیری پارامترها تعریف شده در تابع مطابقت داشته باشد. بعد از ارسال مقادیر ۱۰ و ۵ به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه تابع (خط ۳) دو مقدار با هم جمع می‌شوند و نتیجه به تابع فراخوان (تابعی که تابع calculateSum() را فراخوانی می‌کند) ارسال می‌شود.

در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط ۳) در خط ۹-۶ از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط ۱۱ تابع calculateSum() را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل تابع با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از تابع به وسیله تابع print() نمایش داده می‌شود (خط ۱۱). در برنامه زیر یک تابع تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```
1  func showMessageAndNumber(message: String, number: Int)
2  {
3      print(message)
4      print("Number = \(number)")
5  }
6
7  showMessageAndNumber(message:"Hello World!", number:100)
```

```
Hello World!
Number = 100
```

در مثال بالا یک تابع تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع Int دریافت می‌کند. تابع به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۷ تابع را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر تابع به یکی از دو صورت زیر فراخوانی می‌شد :

```
showMessageAndNumber(message:100, number:"Hello World!")
showMessageAndNumber(number:100, message:"Hello World!")
```

در برنامه خطا به وجود می‌آمد. چون در حالت اول، عدد ۱۰۰ به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شد. و در حالت دوم هم ترتیب ارسال آرگومان ها رعایت نشده است. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی تابع، از لحاظ نوع و پس و پیش بودن، مهم است.

ممکن است که این سوال برایتان پیش آمده باشد که آیا هیچ راهی برای ارسال آرگومان ها، بدون رعایت ترتیب (نه نوع) آنها وجود ندارد؟ جواب این است که، امکان ارسال بدون رعایت ترتیب وجود دارد ولی آن هم با شرایطی خاص. به مثال ابتدای درس بر می گردیم و خط ۱ آن را به صورت زیر تغییر می دهیم:

```
func calculateSum(_ number1: Int, _ number2: Int) -> Int
```

همانطور که مشاهده می کنید، ما قبل از نام پارامترها از علامت _ استفاده کرده ایم. این کار باعث می شود که ما در هنگام فراخوانی تابع، لازم نباشد که نام پارامترها را ذکر کنیم، یعنی می توانیم خط ۱۱ همین مثال را به صورت زیر بنویسیم:

```
print("Sum = ", calculateSum(num1! , num2!))
```

حال چون پارامترهای این تابع هر دو یکسان و از نوع Int هستند، در نتیجه خط بالا را به صورت زیر هم بنویسیم، فرقی نمی کند:

```
print("Sum = ", calculateSum(num2! , num1!))
```

در آن مثال دو عدد از نوع Int به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هر دو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای تابع دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
func showPersonStats(age: Int, height: Int)
{
    print("Age = \(age)")
    print("Height = \(height)")
}

//Using the proper order of arguments
showPersonStats(age: 20, height:160)

//Acceptable, but produces odd results
showPersonStats(height: 160, age: 20)
```

در مثال بالا، نشان داده شده است که حتی اگر تابع دو آرگومان با یک نوع داده‌ای قبول کند، باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال، در اولین فراخوانی تابع بالا اشکالی به چشم نمی‌آید، چون سن شخص ۲۰ و قد او ۱۶۰ سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص ۱۶۰ و قد او ۲۰ سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما توابع کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که، شما حتی می‌توانید مقدار برگشتی از یک تابع را به عنوان آرگومان به تابع دیگر، ارسال کنید.

```
func myFunction() -> Int
{
    return 5
}

func anotherFunction(number: Int)
{
    print(number)
}

// Codes skipped for demonstration
anotherFunction(number: myFunction())
```

```
5
```

چون مقدار برگشتی تابع myFunction() عدد ۵ است و به عنوان آرگومان به تابع anotherFunction() ارسال می‌شود، خروجی کد بالا هم عدد ۵ است.

Variadic Functions

Variadic Functions امکان ارسال تعداد دلخواه پارامترهای همنوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. برای ایجاد تابعی که به تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) به صورت زیر استفاده می‌شود :

```
 func functionName (vaiableName :dataType...) -> dataType
{
   ...
}
```

همانطور که در کد بالا مشاهده می کنید، کافیست، آرگومان هایی که تابع قرار است دریافت کن را به صورت vaiableName… :dataType بنویسید. یعنی یک نام، سپس نوع متغیرها و در نهایت علامت ... را ذکر کنید. به مثال زیر توجه کنید:

```
1  func CalculateSum(numbers: Int...) -> Int
2  {
3      var total :Int = 0
4
5      for num in numbers
6      {
7          total += num
8      }
9      return total
10 }
11
12 print("1 + 2 + 3         = ", CalculateSum(numbers: 1, 2, 3))
13 print("1 + 2 + 3 + 4     = ", CalculateSum(numbers: 1, 2, 3, 4))
14 print("1 + 2 + 3 + 4 + 5 = ", CalculateSum(numbers: 1, 2, 3, 4, 5))
```

```
1 + 2 + 3         = 6
1 + 2 + 3 + 4     = 10
1 + 2 + 3 + 4 + 5 = 15
```

همانطور که در مثال بالا مشاهده می‌کنید، یک تابع به نام CalculateSum() در خط ۱ تعریف شده است. برای اینکه این تابع تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) بعد از نوع داده‌ای پارامتر آن استفاده شده است. در اصل کلمه numbers یک آرایه است، که وقتی ما آرگومان‌ها را به تابع ارسال می‌کنیم، در این آرایه ذخیره می‌شوند. حال تابع را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم و سپس با استفاده از حلقه for این آرگومانها را جمع و به تابع فراخوان برگشت می‌دهیم. وقتی از چندین پارامتر در یک تابع استفاده می‌کنید، فقط یکی از آنها باید دارای علامت سه نقطه (...) باشد.

سربارگذاری توابع

سربارگذاری توابع یا Function Overloading، به شما اجازه می‌دهد که، دو یا چند تابع با نام یکسان تعریف کنید که، دارای امضا و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومانهایی که شما به تابع ارسال می‌کنید به صورت خودکار تشخیص می‌دهد که، کدام تابع را فراخوانی کرده‌اید، یا کدام تابع مد نظر شماست. امضای یک تابع نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید :

```
func myFunction(x: Int, y: Double z: String)
```

که امضای تابع بالا

```
myFunction(Int, Double , String)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای تابع نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری توابع آمده است.

```
1  func showMessage(_ number: Double)
2  {
3      print("Double version of the method was called.")
4  }
5
6  func showMessage(_ number: Int)
7  {
8      print("Integer version of the method was called.")
9  }
10
11 showMessage(9.99)
12 showMessage(9)
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو تابع با نام مشابه تعریف شده‌اند. اگر سربارگذاری تابع توسط Swift پشتیبانی نمی‌شد برنامه زمان زیادی برای انتخاب یک تابع از بین توابعی که فراخوانی می‌شوند لازم داشت. رازی در نوع پارامترهای تابع نهفته است. کامپایلر بین دو یا چند تابع در صورتی فرق می‌گذارد که، پارامترهای متفاوتی داشته باشند. وقتی یک تابع را فراخوانی می‌کنیم، تابع نوع آرگومانها را تشخیص می‌دهد. در فراخوانی اول (خط ۱۱) ما یک مقدار Double را به تابع showMessage() ارسال کرده‌ایم در نتیجه تابع showMessage() (خطوط ۱-۴) که دارای پارامتری از نوع Double اجرا می‌شود. در بار دوم که تابع فراخوانی می‌شود (خط ۱۲) ما یک مقدار Int را به تابع showMessage() ارسال می‌کنیم تابع showMessage() (خطوط ۶-۹) که دارای پارامتری از نوع Int است اجرا می‌شود. معنای اصلی سربارگذاری تابع همین است که توضیح داده شد. هدف اصلی از سربارگذاری توابع این است که، بتوان چندین تابع که وظیفه یکسانی انجام می‌دهند، را تعریف کرد.

محدوده متغیر

متغیرها در Swift، دارای محدوده هستند. محدوده یا scope یک متغیر، به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال، متغیری که در داخل یک تابع تعریف می‌شود، فقط در داخل بدنه تابع قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو تابع مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند :

```
1  func demonstrateScope()
2  {
3      let number = 5
4
5      print("number inside function  demonstrateScope() = \(number)")
6  }
7
8
9  var number = 10
10
11 demonstrateScope()
12
13 print("number outside function demonstrateScope() = \(number)")
```

```
number inside function DemonstrateScope() = 5
number outside function DemonstrateScope() = 10
```

مشاهده می‌کنید که، حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که، دارای محدوده‌های متفاوتی باشند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در خط ۱۰ هیچ ارتباطی به متغیر داخل تابع demonstrateScope() در خط ۳ ندارد. وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد.

پارامترهای پیشفرض

پارامترهای پیشفرض همانگونه که از اسمشان پیداست دارای مقادیر پیشفرضی هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. اگر به اینگونه پارامترها، آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید :

```
1  func PrintMessage(_ message: String = "Welcome to Swift Tutorials!")
2  {
3    print(message)
4  }
5
6  PrintMessage()
7  PrintMessage("Learn Swift Today!")
```

تابع PrintMessage() (خطوط ۲-۱) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط ۱). دو بار تابع را فراخوانی می‌کنیم. در اولین فراخوانی (خط ۶) ما آرگومانی به تابع ارسال نمی‌کنیم بنابراین تابع از مقدار پیشفرض (Welcome to Swift Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط ۷) یک پیغام (آرگومان) به تابع ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود. اگر از چندین پارامتر در تابع استفاده می‌کنید همه پارامترهای اختیاری را در آخر بقیه پارامترها ذکر بنویسید. دلیل این امر این است که معمولا پارامترهایی که آرگومان

دریافت می کنند، نسبت به پارامترهای پیشفرض اهمیت بیشتری دارند. در نتیجه برای شفافیت در کدنویسی، عادت کنید که آنها را قبل از پارامترهای با مقدار پیشفرض بنویسید.

پایان ویرایش اول کتاب ۹۷/۱۱/۲۴

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لینک تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com