پردیس علوم
دانشکده ریاضی، آمار و علوم کامپیوتر

آشنایی با زبان برنامه نویسی سوئیفت (Swift)

نگارنده

اسماء عرب

استاد راهنما: مهندس ابراهیم نقیب زاده مشایخ

پروژه برای دریافت درجه کارشناسی
در رشته علوم کامپیوتر

تیر ۱۳۹۸

چکیده

این مقاله قصد معرفی زبان برنامه‌نویسی سوئیفت [۱] را دارد. ویژگی‌های زبان(امنیت، توسعه سریع، متن‌باز، زمین بازی و...) سوئیفت بیان می‌شود و تقاضا به سمت آن و بازار کار آن و نحوه مدیریت حافظه در آن مورد بحث قرار می‌گیرد و این زبان با دیگر زبان‌های برنامه‌نویسی پرکاربرد از جمله، پایتون [۲] ، آبجکتیوسی [۳] ، گو [۴] ، کاتلین [۵] و جاوا [۶] مقایسه می‌شود تا علت رشد و محبوبیت آن نشان داده شود و به برنامه‌نویس کمک می‌کند که متناسب با خواسته خود، زبان برنامه‌نویسی موردنظر را انتخاب کند و برنامه‌ای خوب و دقیق بدست آورد.

به طور خلاصه سوئیفت آموزش داده می‌شود. این زبان بسیار گسترده است و برای یادگیری به منابعی بیشتر از یک مقاله نیاز است. اما در این مقاله تمام تلاش صورت گرفته است تا کاربردهای مهم بیان شود.

[1] Swift
[2] Python
[3] Objective-c
[4] Go
[5] Kotlin
[6] Java

پیشگفتار

سوئیفت یک زبان برنامه‌نویسی چند شیوه‌ای و از نوع کامپایلری است که برای توسعه‌ی واچ‌اواس [۷] ، مک‌اواس [۸] ، آی‌اواس [۹] و تی‌وی‌اواس [۱۰] توسط شرکت اپل ساخته شده است. سوئیفت برای کار با چارچوب های [۱۱] کوکا [۱۲] و کوکا تاچ [۱۳] اپل و تعامل با حجم عظیمی از کدهای آبجکتیوسی نوشته شده، برای محصولات اپل طراحی شده است و قصد دارد خطاهای برنامه‌نویسی را کمتر کند و امنیت بیشتری نسبت به آبجکتیوسی و در عین حال نگارش مختصرتر و کوتاه‌تری داشته باشد.
این زبان توسط کامپایلر LLVM که درون برنامه‌ی ایکس‌کد [۱۴] قرار دارد (و برای لینوکس [۱۵] به صورت برنامه‌ی جدا در دسترس است) ساخته می‌شود و سپس از زمان اجرا [۱۶] زبان آبجکتیوسی بهره می‌برد که اجازه می‌دهد کدهای سوئیفت با کدهای سی، سی‌پلاس‌پلاس و آبجکتیوسی در کنار هم در یک برنامه اجرا شوند.

توسعه‌ی سوئیفت در سال ۲۰۱۰ توسط کریس لاتنر [۱۷] آغاز شد که با همکاری برنامه‌نویسان اپل ادامه پیدا کرد. سوئیفت از زبان‌های آبجکتیوسی، سی‌ال‌یو [۱۸] ، سی [۱۹] ، پایتون، رابی [۲۰] ، راست [۲۱] و بسیاری دیگر از زبان‌ها الهام گرفته شده است. در ۲ ژوئن ۲۰۱۴، برنامه‌ی موبایل کنفرانس جهانی توسعه‌دهندگان[۲۲]، به‌عنوان اولین برنامه به زبان سوئیفت نوشته و منتشر شد. این

[7] watchOS
[8] macOS
[9] iOS
[10] tvOS
[11] framwork
[12] Cocoa
[13] Cocoa Touch
[14] Xcode
[15] Linux
[16] runtime
[17] Chris Lanther
[18] CLU
[19] C
[20] Ruby
[21] Rust
[22] WWDC

زبان، در نظرسنجی توسعه‌دهندگان در سال ۲۰۱۵ ، رتبه اول محبوب‌ترین زبان برنامه‌نویسی را کسب کرد و در سال ۲۰۱۶ رتبه دوم را به خود اختصاص داد .

بخش اول پروژه، مفاهیم مقدماتی زبان سوئیفت بیان شده‌است. در بخش دوم، نحوهای [۲۳] موردنیاز برای برنامه‌نویسی به این زبان پرداخته است و در بخش سوم به مقایسه آن با زبان‌های دیگر پرداخته است.[۱]

[23] syntax

فهرست مطالب

۱ مفاهیم مقدماتی **ج**

۱.۱ محیط برنامه‌نویسی زبان سوئیفت ۱

۲.۱ ویژگی‌های زبان سوئیفت ۱

۱.۲.۱ نقاط قوت سوئیفت ۱

۲.۲.۱ نقاط ضعف سوئیفت ۴

۳.۱ آینده و بازارکار سوئیفت ۴

۴.۱ مدیریت حافظه در سوئیفت ۵

۵.۱ تایپ‌ها، متغیرها و هدف‌گذاری ۶

۶.۱ انواع مقادیر ۷

۲ برنامه‌نویسی به زبان سوئیفت **۹**

۱.۲ متغیرها ۱۰

۲.۲ اختیاری‌ها ۱۰

۳.۲ ثابت‌ها ۱۱

۴.۲ عبارات ریاضی ۱۱

۵.۲ آرایه‌ها ۱۲

۶.۲ توضیحات ۱۲

۷.۲ دیکشنری‌ها ۱۳

۸.۲ حلقه تکرار For-in ۱۳

۹.۲ حلقه‌های تکرار While , Do-While ۱۴

۱۰.۲ سوییچ ۱۵

۱۱.۲ مفهوم شئ‌گرایی و کلاس‌ها ۱۶

۱۲.۲ متدها ۱۷

۱۳.۲ پراپرتی ۱۸

۱۴.۲ مفهوم وراثت ۱۹

۱۵.۲ نامریشن و استراکچر ۲۱

۱۶.۲ پروتکل ۲۳

۳ مقایسه سوئیفت با زبان‌های دیگر **۲۴**
۱.۳ مقایسه با زبان پایتون ۲۵
۲.۳ مقایسه با آبجکتیوسی ۲۵
۳.۳ مقایسه با گو ۲۷
۴.۳ مقایسه با کاتلین ۲۷
۵.۳ مقایسه با جاوا ۲۸

۴ نتیجه گیری **۲۹**

فصل ۱

مفاهیم مقدماتی

۱.۱ محیط برنامه‌نویسی زبان سوئیفت

محیط ایکس‌کد یکی از پرطرفدارترین محیط برنامه‌نویسی آی‌اواس است و نرم‌افزار برنامه‌نویسی اپل برای هر دو سیستم‌عامل(آی‌اواس و مک) ، ایکس‌کد می‌باشد. محیط برنامه‌نویسی برای سوئیفت ایکس‌کد است که قبل از هر کاری باید آن را نصب کرد. برای برنامه‌نویسی آن باید دستگاه‌های اپل با سیستم‌عامل مکینتاش [۱] باشد یا روی سیستم‌های ویندوز به صورت مجازی ، وی ام ور [۲] یا ویرچوآل‌باکس [۳]، مک را نصب کرد .[۲]

۲.۱ ویژگی‌های زبان سوئیفت

زبان برنامه‌نویسی سوئیفت جایگزینی برای زبان آبجکتیوسی است که از مفاهیم نظریه زبان برنامه‌نویسی مدرن استفاده می‌کند و سعی دارد یک قاعده‌ی ساده‌تر ارائه کند. سوئیفت در هنگام معرفی شدن به‌عنوان" آبجکتیوسی بدون سی" معرفی شد. زبان برنامه‌نویسی سوئیفت به طور پیش‌فرض اشاره‌گرها و دیگر دسترسی‌های ناامن را افشا نمی‌کند، برخلاف آبجکتیوسی که به طور گسترده از اشاره‌گرها برای اشاره به نمونه‌های اشیاء استفاده می‌کند.
همچنین استفاده آبجکتیوسی از یک قاعده شبیه به اسمال‌تاک [۴] برای فراخوانی متدها با یک سبک نمادنقطه [۵] و سیستم فضای نام[۶] جایگزین شده است که بیشتر برای برنامه‌نویس‌های دیگر زبان‌های رایج شئ‌محور مانند جاوا یا سی آشنا است. سوئیفت پارامترهای نامگذاری شده حقیقی را معرفی کرده است و مفاهیم کلیدی آبجکتیوسی از جمله پروتکل‌ها، کلوژرها و دسته‌بندی‌ها را حفظ کرده است و اغلب قوانین پیشین را با نسخه‌های تمیزتر جایگزین کرده است و اجازه می‌دهد این مفاهیم روی دیگر ساختارهای زبانی اعمال شوند، مانند تایپ‌های شمارشی.
در ادامه به دیگر ویژگی‌های این زبان پرداخته شده است. [۹]

۱.۲.۱ نقاط قوت سوئیفت

سوئیفت زبان برنامه‌نویسی جدید اپل است که در اصل برای توسعه آی‌اواس ایجاد شده است و از مفاهیم پایه‌ای پشتیبانی می‌کند که زبان آبجکتیوسی را انعطاف‌پذیر (مخصوصاً در dynamic dispath و late binding) و قابل گسترش نموده است. این امکانات به عنوان تعادل کارایی و امنیت معروف هستند و سوئیفت برای حل آن طراحی شده است.[۹]

[1] Macintosh
[2] vmware
[3] virtualbox
[4] Smalltalk
[5] dot notation
[6] Namespace

• سوئیفت امن‌تر از زبان‌های برنامه‌نویسی دیگر است.

سوئیفت، برای ایجاد امنیت، یک سیستم معرفی کرده است که به رفع خطاهای رایج برنامه‌نویسی مانند اشاره‌گرهای خالی، کمک می‌کند، همچنین نگارش‌های ساده‌تر برای سهولت در خواندن کد نیز معرفی شده است. برای مشکلات کارایی، اپل تلاش قابل‌توجهی برای بهینه‌سازی انجام داده است که سربار فراخوانی متدها را از بین می‌برد و فرآیند را ساده‌تر می‌کند. اساساً سوئیفت مفهوم گسترش قراردادی [۷]را دربرمی‌گیرد که به تایپ‌ها، ساختارها و کلاس‌ها می‌توان افزود. اپل این قابلیت را به عنوان یک تغییر جدی در شیوه‌ی برنامه‌نویسی می‌داند و به‌عنوان« برنامه‌نویسی قرارداد محور» از آن یاد می‌کند.

مدیریت بهتر حافظه در سوئیفت، فرصت کمتری برای از بین رفتن داده‌ها ایجاد می‌کند. همچنین دسترسی به قسمت‌های اشتباه و امکان تغییر آن‌ها باعث می‌شود امکان اشتباه در برنامه‌نویسی کمتر شود و این که هیچ خطایی باعث نمی‌شود تا بقیه داده‌ها دچار مشکل شوند. همچنین سوئیفت با توجه تایپ ثابت و استفاده از گزینه‌ها، امن‌تر است. اگر کد نیاز به یک رشته داشته باشد، ویژگی‌های سوئیفت تضمین می کند که کد شما یک رشته است نه یک نوع دیگری است .

• توسعه‌ی سریع با سوئیفت.

هنگامی که اپل زبان برنامه‌نویسی سوئیفت را طراحی می‌کرد، دو دغدغه مهم وجود داشت؛ یادگیری آسان و امکان توسعه آسان‌تر و سریع‌تر برای ارتقاء برنامه‌ها. در مقایسه با برنامه آبجکتیوسی مشخص است که اپل موفق شد.

سوئیفت دارای تمامی امکانات یک زبان برنامه‌نویسی مدرن است. از ویژگی‌های مهم آن می‌توان به موارد زیر اشاره کرد :

ـ بدون متغیرهای نامشخص و غیرقابل شناسایی
ـ بدون خطای آرایه‌ای خارج از محدوده
ـ بدون خطاهای بیش از حد
ـ رسیدگی صحیح از طریق مقادیر خالی
ـ مدیریت حافظه خودکار

بنابراین زمان بیشتری صرف نوشتن منطق برای کسب‌وکار واقعی می‌شود و همزمان کمترین زمان ممکن را صرف نوشتن کدها و آرایه ها خواهد شد. علاوه بر این سوئیفت از بیشتر قواعد برنامه‌نویسی دست‌وپاگیر در آبجکتیوسی خود را رها کرده است و در واقع زمان کمتری صرف نوشتن کدهایی می‌شود که همان کار را در آبجکتیوسی انجام می‌دهند. می‌توان به یقین گفت سوئیفت صرفه‌جویی در زمان است.

[7]protocol extensibility

• سوئیفت رایگان و با متن باز است.

یک سال پس از انتشار سوئیفت، اپل آن را به یک متن باز [8] تبدیل کرد. این مسئله غیرمعمول نیست، اما برای شرکتی که بیشتر برنامه هایش براساس این نوشته می‌شد، این عمل فرآیند مهمی محسوب می‌شد.
همچنین اپل به توسعه و بهینه کردن سوئیفت پرداخت. این یک تلاش مهم بود و به کاربران این اجازه را می‌داد تا بتوانند به طور منظم عیب‌یابی و رفع اشکال کنند و قابلیت‌های جدیدی را به یک زبان برنامه‌نویسی اضافه کنند.

• سوئیفت سریع است.

با وجود اینکه سوئیفت یک زبان سطح بالا متمرکز بر توسعه سریع است، خنده‌دار است که کمترین زمان را برای بیشترین میزان توسعه برنامه صرف می‌کند. به گفته اپل، سوئیفت 2.6 برابر سریع‌تر از آبجکتیوسی و 8.4 برابر سریع‌تر از زبان برنامه‌نویسی پایتون 2.7 است. امروزه توسعه‌دهندگان نرم‌افزارها به زبانی نیاز دارند تا از سی‌پلاس‌پلاس که یکی از سریع‌ترین زبان‌های برنامه‌نویسی است، سریع‌تر باشد.
سوئیفت نه‌تنها سریع است، بلکه بسیار قدرتمند است و ویژگی‌های سوئیفت، برنامه‌نویس را قادر به نوشتن کدهای پیشرفته در کمترین زمان می‌کند.

• یادگیری سوئیفت آسان است .

اپل زبان خود را برای سهولت استفاده و سادگی و تطبیق با پایتون ساخته است. نوشتن کد به این زبان بسیار ساده و بی‌تکلف است و این یعنی می‌توان سریع‌تر از هر قالب [9] دیگری این زبان را یاد گرفت.

• سوئیفت محیط تجربه و تست است .

اپل یک حالت زمین بازی [10] را در سوئیفت تعبیه کرده است که این حالت، قابلیتی است که به برنامه‌نویس‌ها در محیط کاربری ایکس‌کد، اجازه می‌دهد تا نتیجه کد را همزمان مشاهده کنند. برنامه‌نویس می‌تواند از زمین بازی برای کار روی ایده‌های برنامه استفاده کند و به این دلیل که از

[8] Open Source
[9] Platform
[10] Playground

برنامه‌ی واقعی‌اش جدا است، براحتی می‌تواند ایده‌های غیرقابل قبول را رها کند.

۲.۲.۱ نقاط ضعف سوئیفت

سوئیفت یک زبان جوان است و با آن برخی تغییر می‌کند. مهاجرت بین نسخه‌ها یک مشکل است. دلیل این مهاجرت این واقعیت است که سوئیفت ABI پایدار نیست، به این معنا است که نسخه‌های جدید سوئیفت نمی‌توانند با نسخه‌های قدیمی آن کار کنند . در واقع بدان معنا است که زبان را نمی‌توان با سیستم‌عامل بسته‌بندی کرد. این یک معامله بزرگ برای شرکت‌هایی با برنامه‌های بزرگ است که بشدت با حجم برنامه مبارزه می‌کنند زیرا سوئیفت همراه با برنامه و افزایش اندازه می‌باشد.

مسئله دیگر این است که سوئیفت با ایکس‌کد خوب بازی نمی کند. هنگام کار با سوئیفت احساس ناآرامی زیاد دارد و گاهی اوقات کار خودکار انجام نمی‌شود. این عجیب است که چگونه اپل سخت سوئیفت را تحت فشار قرار داده است.

سوئیفت همچنین دارای مشکلات مربوط به پردازش رشته است. بیشتر این موارد در طی مصاحبه‌ها اتفاق می‌افتد. متأسفانه برای متخصصان سوئیفت، مصاحبه‌گران دوست دارند سوالاتی را مطرح کنند که شامل دستکاری رشته هستند. این امر به وسیله این واقعیت که راه رشته‌ها در حال پردازش است بین نسخه‌های سوئیفت تغییر کرده است . [۹]

۳.۱ آینده و بازارکار سوئیفت

براساس گزارش گیت‌هاب اکتورز در سال ۲۰۱۷، سوئیفت فعال‌ترین زبان برنامه‌نویسی منبع باز در همه پروژه‌ها است. مهم‌تر از آن، تقاضای کاربران برای استفاده از سوئیفت رو به رشد بوده است.

تی‌ان‌دبلیو نیز گزارش داد بر اساس آمارهای ارائه شده، در طول سال ۲۰۱۶ سوئیفت رشدی در حدود ۶۰۰ درصد داشته است. تا پایان سال ۲۰۱۶ آپ‌ورک گزارش کرد که سوئیفت دومین برنامه سریع و منبع باز است و در بررسی‌هایی که استک در سال ۲۰۱۷ انجام داد، سوئیفت چهارمین برنامه محبوب کاربران فعال شده است.

توسعه برنامه ها در حال حاضر بازار داغی دارد. اگر تصمیم دارید که در این زمینه فعال و حرفه‌ای باشید، نیاز دارید که سوئیفت را یاد بگیرید. با اضافه کردن سوئیفت به رزومه خود شانس بیشتری برای رقابت در بازارکار خواهید داشت. مشاغل موجود نسبت به تعداد توسعه‌دهندگان این برنامه بیشتر است. این یعنی اگر کسی تمایل به ساخت برنامه [11] با سوئیفت داشته باشد، فرصت‌های زیادی برای امرارمعاش از طریق این مهارت خواهد داشت ودر حال حاضر نیاز به توسعه‌دهندگان

[11] Application

سوئیفت احساس می‌شود.[۹]

۴.۱ مدیریت حافظه در سوئیفت

زبان برنامه‌نویسی سوئیفت از شمارش مرجع اتوماتیک (ARC) برای مدیریت حافظه استفاده می‌کند. اپل در گذشته نیاز به مدیریت حافظه دستی در آبجکتیوسی داشت، اما ARC را در سال ۲۰۱۱ معرفی کرد که اجازه تخصیص و آزادسازی راحت‌تر حافظه را می‌دهد. یکی از مشکلات ARC، احتمال ایجاد یک چرخه مرجع قوی است که نمونه‌های دو کلاس مختلف هر کدام به دیگری ارجاع دارند که باعث می‌شود در حافظه نشت کنند، زیرا هرگز آزاد نمی‌شود.

زبان برنامه‌نویسی سوئیفت کلمات کلیدی weak و unowned را فراهم کرده است تا از چرخه‌های مرجع قوی خودکاری شود. عموماً یک رابطه پدر – فرزندی از مرجع قوی استفاده می‌کند، درحالی‌که یک رابطه فرزند – پدری یا از یک مرجع weak استفاده می‌کند که عناصر پدر و فرزند می‌توانند با هم نامرتبط باشند، یا از unowned استفاده می‌کند که یک فرزند همیشه یک پدر دارد، اما پدر ممکن است فرزندی نداشته باشد. مرجع های weak باید متغیرهای اختیاری باشند، زیرا می‌توانند تغییر کنند و nil شوند. یک بسته [۱۲] درون یک کلاس نیز می‌تواند با گرفتن ارجاعات به خود، یک چرخه مرجع قوی ایجاد کند. ارجاعات به خود را می توان با استفاده از یک لیست ضبط [۱۳] به عنوان weak یا unowned در نظر گرفت.[۹]

[12] closure
[13] Capture List

۵.۱ تایپ‌ها، متغیرها و هدف‌گذاری

در محیطهای کوکا و کوکا تاچ ،بسیاری از کلاس‌های رایج بخشی از کتابخانه Foundation Kit بودند. آن‌ها شامل کتابخانه رشته NSString و کلاس‌های مجموعه NSArray و NSDictionary بودند. زبان آبجکتیوسی تکه‌های مختلفی از Synthetic Sugar عرضه می‌کرد تا اجازه دهد برخی از این اشیاء به صورت لحظه‌ای در زبان ساخته شوند، اما وقتی ساخته شدند، اشیاء با فراخوانی شئ دستکاری می‌شدند. به‌عنوان مثال، الحاق کردن دو عدد NSStrings نیازمند فراخوانی‌های متدی مانند زیر بود:

```
NSString*
str
=
@
”hello”
;
str
=
[
str
stringByAppendingString
;
”world”
]
;
```

در زبان برنامه‌نویسی سوئیفت بسیاری از این تایپ‌های اساسی به هسته زبان اضافه شده‌اند و می‌توان آن‌ها را به صورت مستقیم دستکاری کرد. به‌عنوان مثال، رشته‌ها به صورت نامرئی به NSString متصل شده‌اند و اکنون می‌توان با عملگر + آن‌ها را الحاق کرد که این امر قاعده نحوی بسیار راحت‌تر شود. مثال قبل به شکل زیر نوشته می‌شود:

```
var
str
=
”hello”
str
+=
”world”
```

زبان برنامه‌نویسی سوئیفت از پنج سطح کنترل دسترسی برای علائم، پشتیبانی می‌کند:
public ، open ، internal ، private و fileprivate . برخلاف بسیاری از زبان‌های شئ‌گرا، این کنترل‌های دسترسی سلسله مراتب وراثت را نادیده می‌گیرند. private نشان می‌دهد که یک

نشانه تنها از طریق محدوده کنونی قابل دسترسی است، fileprivate نشان می‌دهد که تنها از داخل فایل قابل دسترسی است، internal نشان می‌دهد که از داخل ماژول قابل دسترسی است، public نشان می‌دهد که از هر ماژولی قابل دسترسی است و open (تنها برای کلاس‌ها و متدهای آن‌ها) نشان می‌دهد که ممکن است کلاس در خارج از ماژول به زیرکلاس تبدیل شده باشد.

۶.۱ انواع مقادیر

در بسیاری از زبان‌های شئ‌گرا، اشیا به صورت داخلی در دو بخش ارائه می‌شوند. شئ به صورت یک بلوک داده ذخیره شده و در حافظه هیپ [۱۴] قرار گرفته است، در حالی که نام آن شئ توسط یک نشانگر نمایش داده می‌شود. اشیاء با کپی کردن مقدار نشانگر بین متدها ارسال می‌شوند و به همان داده در هیپ اجازه می‌دهند در دسترس هر کسی که یک کپی دارد، قرار بگیرد. برعکس، تا تایپ‌های اصلی مانند اعداد صحیح و اعداد با ممیز شناور به صورت مستقیم ارائه می‌شوند. داده در هندل وجود دارد، نه در نشانگر و آن داده به صورت مستقیم با کپی شدن به متدها ارسال می‌شود. به هر دو سبک دسترسی" انتقال با مقدار" برای تایپ‌های اصلی گفته می‌شود.
هر دو مفهوم دارای مزایا و معایبی هستند. اشیاء زمانی مفید هستند که داده بزرگ است، مانند توضیح یک پنجره یا محتوای یک سند. در این موارد، دسترسی به آن داده با کپی کردن یک مقدار چهار بایتی یا هشت بایتی فراهم می‌شود، نه با کپی کردن یک ساختار داده کامل. هر چند مقادیر کوچکتر مانند اعداد صحیح با نشانگرها هم اندازه هستند، بنابراین مزیتی در انتقال دادن یک نشانگر به جای مقدار وجود ندارد. همچنین انتقال از طریق مرجع نیاز به یک عملیات نابودی [۱۵] دارد که می‌تواندر بعضی عملیات‌ها سربار زیادی توضیح کند، معمولاً آن‌هایی که این نوع‌های مقداری اصلی را دارند، مانند عملیات ریاضیاتی.

زبان برنامه‌نویسی سوئیفت هم مانند زبان سی و برخلاف بسیاری دیگر از زبان‌های شئ‌گرا، پشتیبانی داخلی از اشیاء با استفاده از قواعد انتقال از طریق مرجع یا انتقال از طریق مقدار دارد. اولی از اعلان class

و دومی از اعلان استراکت [۱۶] استفاده می‌کند. استراکت‌ها در سوئیفت تقریباً تمام ویژگی‌های هاclass را دارند: متدها، استفاده از پروتکل‌ها و استفاده از مکانیزهای گسترش. به این دلیل شرکت اپل به جای اشیاء یا مقادیر، معمولاً همه داده‌ها را نمونه [۱۷] می‌نامد. البته استراکت‌ها از وراثت پشتیبانی نمی‌کنند.
برنامه‌نویس می‌تواند با آزادی تصمیم بگیرد که کدام قواعد برای هر ساختار داده در برنامه مناسب است. ساختارهای بزرگتر مانند پنجره‌ها به‌عنوان class تعریف می‌شوند تا بتوانند به‌عنوان نشانگرها

[14] Heap
[15] dereferencing
[16] Struct
[17] Instance

منتقل شوند. ساختار های کوچکتر مانند یک نقطه دوبعدی را می‌توان به‌عنوان یک استراکت تعریف کرد که توسط مقدار منتقل می‌شود و اجازه دسترسی مستقیم به داده‌های داخلی‌شان را بدون نیاز به نابودی می‌دهند. بهبود عملکرد مفهوم انتقال با مقدار به گونه گونه‌ای است که سوئیفت از این تایپ‌ها تقریباً برای تمام انواع داده‌های عمومی استفاده می‌کند از جمله int و double و تایپ‌هایی که معمولاً توسط اشیاء نمایش داده می‌شوند، مانند string و .Array استفاده از انواع مقداری همچنین می‌تواند منجر به بهبود عملکرد چشم‌گیری در برنامه‌های کاربردی شود.

برای اطمینان از این که حتی بزرگترین استراکت‌ها باعث کاهش کارایی نمی‌شوند، سوئیفت از روش اشتراک گذاری ضمنی استفاده می‌کند تا اشیاء تنها زمانی کپی شوند که برنامه سعی دارد مقداری را در آن‌ها تغییر دهد. یعنی اکسسورهای [۱۸] مختلف در واقع یک نشانگر به همان منبع داده دارند، اما این بسیار پایین‌تر از سطح زبانی و در واحد مدیریت حافظه کامپیوتر [۱۹] صورت می‌گیرد. بنابراین با این که داده به صورت فیزیکی به صورت یک نمونه در حافظه ذخیره شده است، در سطح برنامه این مقادیر مجزا هستند و جداسازی فیزیکی تنها در صورت نیاز توسط اشتراک گذاری ضمنی اجرا می‌شود.[۹]

[18] Accessor
[19] MMU

فصل ۲

برنامه‌نویسی به زبان سوئیفت

۱.۲ متغیرها

یک متغیر [1] ، یک مکان در حافظه است که دارای یک نام نمادین با عنوان شناسه می‌باشد و می‌تواند حاوی مقدار باشد. بهتر است متغیر را ظرفی تصور کنیم که دارای یک نام می‌باشد و قادر است مقداری اطلاعات را نگهداری کند. برای کار با متغیرها همیشه نیاز به ایجاد آن‌ها داریم.
در زبان سوئیفت ایجاد متغیرها ساده است و از کلیدواژه‌ی var و نام متغیر استفاده می‌شود، به‌عنوان مثال:

```
var str = "salam"
```

در مثال بالا ابتدا کلیدواژه‌ی var و سپس نام str ، برای این متغییر تعیین شده است. عبارت بعد از مساوی همان مقداری خواهد بود که در حافظه ذخیره می‌شود. مقدار متغیر یا اطلاعاتی که متغیر نگهداری می‌کند، می‌تواند رشته یا عدد باشد. رشته به دنباله‌ای از کاراکترها گفته می‌شود، بنابراین هر کلمه و حتی هر متن یک رشته به شمار می‌آید، رشته‌ها در اکثر زبان‌های متعارف برنامه‌نویسی استرینگ نامیده می‌شوند. در زبان سوئیفت تعیین نوع متغیر اجباری نیست ولی بهتر است انجام شود، به‌عنوان مثال:

```
var str:String = "salam"
```

که مانند بالا است ولی بعد از نام متغیر، نوع آن را معرفی کرده است.[۲، ۳]

۲.۲ اختیاری‌ها

برای کار با متغیرهایی که ممکن است مقدار داشته یا نداشته باشد، می توان از اختیاری‌ها [2] استفاده کرد که متغیری است که یا مقداری دارد و یا به نشانه‌ی تهی بودن حاوی nil می‌باشد، به عبارتی دیگر یا مقدار دارد یا از مقدار تهی می‌باشد. جهت اعلان یک متغیر به‌عنوان اختیاری، کافی است یک علامت "؟" پس از نوع مقدار تایپ کرد. به‌عنوان مثال:

```
let optionalInt: Int? = 9
```

برای استخراج و دسترسی به مقدار اصلی و اولیه‌ی یک متغیر اختیاری، لازم است آن را unwrap کرد. به طور خلاصه باید گفت که برای دسترسی به مقدار اختیاری، از عملگر "!" استفاده می‌شود. لازم به ذکر است که تنها زمانی باید از عملگر مزبور استفاده کرد که این اطمینان باشد که مقدار اصلی یا زیرین nil نیست. مانند زیر:

```
let actualInt: Int = optionalInt!
```

[1] Variable
[2] Optional

اختیاری‌ها در سوئیفت کاربرد زیادی دارند و در مواقعی که ممکن است مقداری وجود داشته یا نداشته باشد، می‌توانند فوق‌العاده مفید واقع شوند. از آن‌ها می‌توان به خصوص برای تبدیل نوع به صورت آزمایشی بهره گرفت.[۳]

۳.۲ ثابت‌ها

در تمامی زبان‌های برنامه‌نویسی علاوه بر متغیرها، که می‌توان مقدار درونی آن‌ها را به اقتضای کاربردشان در هر جای برنامه تغییر داد، نوع دیگری نیز به نام ثابت [۳] وجود دارد. همانطور که از اسم آن پیداست مقدار درونی ثابت در تمامی طول برنامه ثابت خواهد ماند. برای ایجاد ثابت‌ها باید از کلیدواژه‌ی Let استفاده کرد، به‌عنوان مثال:

```
Let myConstant:Int = 123
```

ایجاد ثابت‌ها، می‌تواند با تعیین نوع همراه باشد و هم بدون آن، البته به دلیل بهینگی در میزان مصرف حافظه برای متغیرها، یا ثابت‌ها بهتر است نوع آن‌ها توسط برنامه‌نویس مشخص گردد.[۲]

۴.۲ عبارات ریاضی

همه‌ی شما با عبارات ریاضی یا همان چهار عمل اصلی آشنایی دارید ولی چگونگی استفاده از آن‌ها در زبان‌های برنامه‌نویسی کمی متفاوت است. روی متغیرهای عددی می‌توان چهار عمل اصلی(بترتیب اولویت ضرب، تقسیم ،جمع و تفریق) ریاضی را پیاده‌سازی کرد و در متغیری دیگر ذخیره کرد، به‌عنوان مثال:[۲]

```
var B:Int = 30 * 2
var C:Int = B/15
var A:Int = 10 + 20
var D:Int = A - B
```

[3] Constant

۵.۲ آرایه‌ها

آرایه [۴] از جمله کاربردی‌ترین متغیرهای تمامی زبان‌های برنامه‌نویسی است. در تعریف‌های متعارف، آرایه به این صورت تفسیر شده است:
آرایه لیست متناهی از عناصر داده‌ای هم‌نوع است. انواع آرایه وجود دارد: آرایه‌های یک‌بعدی و آرایه‌های دوبعدی. تمامی متغیرها با کلیدواژه‌ی var یا Let (برای ثابت‌ها) ایجاد می‌شوند. آرایه نیز متغیری توسعه یافته است، پس از این قانون مستثنی نیست. به‌عنوان مثال:

```
var Name[String] = ["Sam","Dam"]
```

در این مثال آرایه‌ای به نام Name ساخته‌ایم که در آن رشته‌ای از استرینگ‌ها، خانه‌های آرایه را پر کرده است. برای اضافه کردن شخصی دیگر به اسامی بالا به صورت زیر عمل می‌کنیم:

```
Name +=["Amin"]
```

این نام به انتهای آرایه اضافه خواهد شد.
در تمامی زبان‌های برنامه‌نویسی شئ‌گرا، اکثر اشیاء مثل آرایه‌ها، متغیرها، ثابت‌ها و...دارای متدهایی هستند که کار با آن‌ها را ساده تر می‌کند. از جمله مهمترین و کاربردی‌ترین متد برای آرایه‌ها، تابع append() نام دارد که می‌توان از آن برای اضافه کردن رشته‌ای به آرایه استفاده کرد، به‌عنوان مثال:

```
Name. append("Amin ")
```

حال برای نسبت دادن مقداری از آرایه به یک متغیر، با استفاده از اندیس آرایه انجام می‌پذیرد، به‌عنوان مثال:

```
var Array :[String] = ["Dog" , "Cat"]
var Animal : String = Array[0]
```

در این مثال، مقدار خانه صفر آرایه ،یعنی Dog درون متغیر Animal ریخته شده است. [۲، ۴]

۶.۲ توضیحات

توضیحات [۵] یک قطعه متن در کد برنامه است که به‌عنوان بخشی از برنامه ترجمه نمی‌شود اما توضیحاتی و اطلاعات مفیدی را درباره‌ی بخش‌های مختلف کد ارائه می‌دهد. کامنت‌ها به دو دسته تقسیم می‌شوند: ۱. توضیحات تک خطی که پس از // درج می‌شود ۲. توضیحات چند خطی که بین */ و /* قرار می‌گیرند.[۲]

[4] Array
[5] comments

۷.۲ دیکشنری‌ها

دیکشنری‌ها [۶] را می‌توان نوع دیگری از آرایه‌ها دانست. شباهت دیکشنری‌ها به آرایه‌های دوبعدی در تعریف کلی آن است، اما تفاوت‌هایی اساسی وجود دارد که وجود دیکشنری‌ها را لازم و پررنگ می‌کند.
ساختار دیکشنری‌ها به صورتی است که به جای اندیس‌های عددی، می‌توان اندیس‌های متنی برای آن‌ها انتخاب کرد. در آرایه‌ها، اندیس‌ها، شماره‌ی خانه‌های حافظه را تشکیل می‌دادند و مقادیر، مقدار درون هر خانه حافظه نام داشت اما در دیکشنری، کلید نام هر خانه حافظه می‌باشد و مقادیر همانند قبل، مقدار درون هر خانه حافظه نام دارد، به‌عنوان مثال:

```
var myfriends : [String : Int] = ["Ali":35 ,"Hamid":23 , "Reza":27]
```

در مثال بالا دیکشنری ایجاد شد که از نام‌ها به عنوان کلیدها و از مقادیر به عنوان سن استفاده شده است.[۲]

۸.۲ حلقه تکرار For-in

حلقه‌ها بسیار پر کاربرد هستند و کاربردهای حلقه‌ی for-in آنقدر زیاد است چرا که در طول آموزش برنامه‌نویسی سوئیفت تقریباً همیشه با آن‌ها سر و کار خواهید داشت. تکرار رویدادی را، تا رسیدن به شرطی خاص ادامه می‌دهد. حلقه‌های تکرار for-in حالت هوشمندی از نوع سنتی حلقه‌ها به شمار می‌روند که در زبان برنامه‌نویسی سوئیفت یک مزیت بزرگ حساب می‌شود، به‌عنوان مثال:

```
for index in 1...5{
    statements
    }
```

خروجی حلقه‌ی بالا ۵ بار تکرار می‌شود. برای پیمایش دیکشنری با حلقه for-in :

```
let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
    for (animalName, legCount) in numberOfLegs {
    print("(animalName)s have (legCount) legs")
    }
```

در مثال بالا ابتدا یک دیکشنری از نام انواع حشرات(کلید) و تعداد پای هر کدام(مقدار) ساخته شده است و همانطور که گفته شد در حلقه‌های For-in برای پیمایش یک آرایه، صرفاً یک شمارنده ساخته شده است (اندیس) ، و خود کامپایلر متوجه می‌شود که این شمارنده(یا در اصل: متغیر) در هر لحظه مقدار کدام سلول از آرایه را نگهداری می‌کند، اما در دیکشنری‌ها به غیر از

[6] Dictionary

مقدار، نیاز به نگهداری کلید هر خانه از دیکشنری را نیز است، پس باید برای هر کدام (کلید و مقدار) یک شمارنده‌ی جدا تعریف کرد، در این مثال animalName مقدار کلید هر خانه، که شامل نام حشره می‌شود را نگهداری می‌کند و legCount اشاره به مقدار دارد، که تعداد پای هر حشره متناظر با نام آن را نگهداری می‌کند.[۲، ۳]

۹.۲ حلقه‌های تکرار While , Do-While

حلقه‌های تکرار while دو نوع است:

- حلقه While: ابتدا شرط را بررسی، سپس دستورات را اجرا می‌کند، تا زمانی که شرط برقرار باشد.
- حلقه Do-While: ابتدا دستورات را اجرا، سپس شرط را برای تکرار دستورات بررسی می‌کند.(حداقل یکبار دستورات اجرا می‌شود)

ساختار While به صورت زیر می‌باشد:

```
While condition {
    statements
    }
```

ساختار Do-While به صورت زیر می‌باشد:[۲]

```
do{
    statements
    }While condition
```

۱۰.۲ سوییچ

ساختار سوییچ [۷] در زبان برنامه‌نویسی سوئیفت از قدرت و امکانات ویژه‌ای برخوردار است. این دستور از انواع داده‌ای و همچنین طیف وسیعی از عملیات مقایسه‌ای پشتیبانی می‌کند. در مثال زیر، دستور سوییچ مقدار متغیر را با مقادیر ،هاcase مقایسه کرده و سپس دستور مرتبط با مقدار منطبق را اجرا می‌کند.

```
let vegetable = "red pepper"
    switch vegetable{
    case "celery":
    let vegetableComment = "Add some raisins and make ants on a log."
    case "cucumber", "watercress":
    let vegetableComment = "That would make a good tea sandwich."
    default:
    let vegetableComment = "Everything tastes good in soup."
    }
```

استفاده از دستور پیش‌فرض [۸] در بیشتر موقعیت‌ها ضروری است. با استفاده از پیش‌فرض، این اطمینان بدست می‌آید که در صورت برقرار نبودن شرط هیچ یک از هاcase (برابر نبودن مقدار داخل پرانتز با مقدار هیچ یک از case ها)، یک دستور در ساختمان switch اجرا می‌گردد. در واقع در ساختار چند انتخابی آن، باید از دستور پیش‌فرض استفاده کرد، مگر اینکه این اطمینان باشد که مقدار داخل پرانتز و مورد مقایسه با حداقل یکی از case ها منطبق است.[۲، ۵]

[7] Switch
[8] defult

۱۱.۲ مفهوم شئ‌گرایی و کلاس‌ها

برنامه‌نویسی شئ‌گرا، مفهوم بسیار کاربردی و مهمی در دنیای برنامه‌نویسی به وجود آورده است، هر چند مفهوم جدیدی نیست اما متأسفانه خیلی از برنامه‌نویسان در درک و بهره‌گیری از این مفهوم عاجز هستند؛ چرا که به کار گیری شئ‌گرایی در عالم برنامه‌نویسی، به درک روابط اشیاء با هم، مربوط می‌شود.

اشیاء در تعریف ، قطعه کدهایی هستند که با استفاده از خصوصیات تعریف شده برای آن‌ها، مسئولیتی را انجام می‌دهند. این اشیاء می‌توانند شامل کلاس‌ها و یا متدها باشند که در صورت فراخوانی هر کدام با توجه به داده‌های ارسالی به آن‌ها، وظایفی را انجام داده‌اند و خروجی این وظایف می‌تواند نتیجه‌ی نهایی برنامه ساخته شده را تشکیل دهد، و روند برنامه‌نویسی را تسهیل کند.

برنامه‌نویسی شئ‌گرا، داده‌ها و متدها را در بسته‌هایی به نام کلاس محصور می‌کند. کلاس‌ها دارای خاصیت کپسوله‌سازی هستند. این بدان معناست که اشیاء می‌دانند که چگونه از طریق رابط‌های تعریف شده، با یکدیگر ارتباط برقرار کنند ولی معمولاً اشیاء نباید از چگونگی پیاده‌سازی اشیاء دیگر مطلع باشند، به عبارت دیگر، جزئیات پیاده‌سازی در داخل خود اشیاء پنهان شده‌اند.
برای تعریف کلاس به صورت زیر عمل می‌کنیم:

```
class       {
    init() {

    }
    }
```

برای ساخت کلاس از کلیدواژه‌ی class استفاده می‌کنیم، سپس نام کلاس را نوشته و کروشه را باز می‌کنیم. متد init() لازمه‌ی ساخت هر کلاس است و دستورات درون آن بلافاصله در زمان فراخوانی کلاس اجرا خواهند شد و کروشه بسته و تمام می‌شود.[۲، ۵]

۱۲.۲ متدها

متدها در زبان برنامه‌نویسی شئ‌گرا بسیار کاربرد دارند و برنامه‌نویسی بدون آن ها تقریباً امکان‌پذیر نیست. متدها، توابعی هستند که با استفاده از داده‌های ارسالی به آن‌ها، همچنین وظایفی که در زمان ساخت به آن‌ها محول شده است، خروجی پردازش شده‌ای را به برنامه‌نویس و یا قطعه کدی دیگر در یک سیستم، ارسال می‌کند. متدها به برنامه‌نویس اجازه می‌دهد که وظایف بزرگ برنامه‌ی خود را با استفاده از آن‌ها خرد و به راحتی پیاده‌سازی کنند و از نوشتن دستورات تکراری در طول برنامه، جلوگیری کنند.
ساخت متدها با استفاده از کلیدواژه‌ی func که مخفف عبارت function است، شروع می شود . به عنوان مثال :

```
func doSomething(){
    print(“function Done”)
    }
```

حال اگر متد، مقدار ورودی می‌گرفت به صورت زیر:

```
Func Name (Input:String) -> Int {
    // some code to execute
    Return 0
    }
```

در این مثال همانند متد قبل ، ابتدا از عبارت func برای ایجاد متد استفاده کرده و پس از آن نام متد آمده است. درون پرانتز عبارت ورودی و همچنین نوع آن نوشته شده است، متغیری به نام این‌پوت از نوع رشته داریم و پس از آن ، نوع خروجی متد آمده است که برای مشخص کردن نوع خروجی باید از عبارت -> استفاده کرد. و در انتهای متد از عبارت ری‌ترن برای ارسال خروجی استفاده کرده است.

همچنین نوع پیچیده تری از متدها وجود دارند که می‌توانند مقداری را برای انجام محاسبات از ورودی دریافت کنند، نوع ورودی از ابتدا و در زمان ساخت متد مشخص می‌شود. ساختار این نوع از متدها در زیر مشاهده می‌شود:

```
func Name (Input:String) {
    // Some Code for Execute with Input
    }
```

کاربرد اصلی و نهایی متدها درون کلاس‌ها تعریف می‌شود. ساخت شئ از یک کلاس به معنی آن است که به آن کلاس در هر نقطه‌ای از برنامه، موجودیت داده شود. این کار برای دسترسی به ویژگی‌ها [۹] همچنین متدهای آن کلاس، الزامی است. پس برای دسترسی به متدها و ویژگی‌های یک

[9] Property

کلاس صرفاً نمی‌توان نام آن کلاس را صدا زد، چرا که باید ابتدا از کلاس مورد نظر یک شئ ساخته شود. فراخوانی متد ها کلاس دیگر به صورت ، اسم شئ و نقطه و نام متد کلاس مورد نظر انجام می‌شود.
در محیط زمین بازی می‌توان متدها را بدون وجود کلاس‌ها ساخت و صدا زد، اما در محیط ایکس‌کد این کار امکان‌پذیر نیست، و نمی‌توان متدها را بدون وجود کلاس‌ها فراخوانی کرد، همانطور که کلاس بدون func نیز معنایی ندارد.
نکته: توجه داشته باشید در پروژه‌هایی که در محیط زمین بازی کار می‌کنید، هرگز عبارت import UIKit را حذف نکنید، چرا که این کار باعث بروز خطا در هنگام اجرا دستورات شما می‌شود.[۲، ۵]

۱۳.۲ پراپرتی

متغیرها عنصری اساسی در برنامه‌نویسی به شمار می‌روند، برای ساخت متدها و ارسال داده به آن، همچنین برای ساخت شئ از یک کلاس و حتی برای نگهداری یک عدد یا یک رشته‌ی ساده به متغیرها نیاز پیدا خواهید کرد، در تمام طول یک پروژه، چندین و چند متغیر و ثابت تعریف می‌کنید.
اما متغیرهایی وجود دارند که جایگاه خاصی برای آن‌ها درون کلاس‌ها تعریف شده است که به آن پراپرتی [۱۰] گفته می‌شود. آن‌ها متغیرهای اختصاصی کلاس‌ها هستند، به صورتی که می‌توان در خارج از کلاس‌ها، و پس از ساخت شئ از کلاس موردنظر به آن‌ها مقادیری را نسبت داد. پس پراپرتی‌ها در تمامی طول پروژه قابل دسترسی خواهند بود و به نوعی متغیرهای عمومی کلاس‌ها می‌باشند.
نکته: از یک کلاس می‌توان چندین شئ ساخت و هر شئ می‌تواند پراپرتی متفاوتی را ذخیره داشته باشد. برای ایجاد یک پراپرتی باید مثل معمول، از کلیدواژه‌ی var یا let (برای ساخت ثابت) استفاده کرد، ولی تفاوتی که پراپرتی‌ها با متغیرهای معمولی دارند، در جایگاه تعریف آن‌ها است، به صورتی که آن‌ها را فقط باید در زیر نام کلاس تعریف کرد، ولاغیر. [۲]

[10] Property

۱۴.۲ مفهوم وراثت

گاهی اوقات، نوشتن چند کلاس، و بالطبع نوشتن چند متد لازم می‌شود، اما با استفاده از مفهوم وراثت می‌توان از نوشتن تکراری کلاس‌ها و متدهای یکسان جلوگیری کرد . همانطور که از قبل گفته شد، مفهوم شئ‌گرایی برای پرهیز از تکرار و نوشتن کدهایی پویا به وجود آمده است، یکی از این مفاهیم که کاربردی اساسی را در برنامه‌نویسی به عهده دارد، وراثت و ساخت ساب‌کلاس می‌باشد. به‌عنوان مثال :

```
class father{
    init(){
    print("New person Object Initialized")
    }
    func talk(){
    print("I want Say SomeThing")
    }
    func walk (){
    print("I'm Walking")
    }
    func run(){
    print("I'm Running")
    }
    }
```

یک کلاس به نام فادر، که شامل خصوصیات بدیهی یک شخص می‌باشد، مثل راه رفتن، حرف زدن ، دویدن و ... در مثال بالا ساخته شده است. اگر در یک کلاس، از متدی درون همان کلاس استفاده شود، باید از کلیدواژه‌ی self استفاده شود. به‌عنوان مثال اگر در متد ()walk، متد ()run فراخوانی شود، باید متد ()walk به صورت زیر تغییر پیدا کند:

```
func walk (){
    print("I'm Walking")
    self.run()
    }
```

برای ساختن کلاس چایلد از کلاس فادر که به آن اصطلاحاً ساب‌کلاس گفته می‌شود، و این کار باعث می‌شود تمامی خصوصیات کلاس فادر(متدها و پراپرتی‌ها) به کلاس چایلد نیز منتقل شود. می‌توان کلاسی نوشت که تمامی متدهای فادر را داشته باشد، اما این بدان معنا نیست که کلاس چایلد تماماً به متدهای فادر وابسته است، این کلاس می‌تواند متدها و پراپرتی‌های خاص خود را داشته باشد. می‌توان همانند قبل متدها را درون بدنه‌ی کلاس اضافه کرد:

```
class child:father{
    func think(){
    print("i'm thinking ... ")
    }
    }
```

نکته: همانطور که دیده شد در کلاس چایلد وجود متد init() اجباری نمی‌باشد. در حقیقت ساب‌کلاس‌ها احتیاجی به متد init() نخواهند داشت ولی اگر نیاز باشد این متد مستقل در کلاس چایلد وجود داشته باشد، باید مثل قبل آن را دوباره‌نویسی(override) کرد، ولی اگر در متد init() چایلد، متد init() فادر را فراخوانی کرد باید با کلیدواژه‌ی ،super فراخوانی انجام شود. به صورت زیر:

```
class child:father{
    override init() {
    super.init()
    }
    func think(){
    print("i'm thinking ... ")
    super.talk()
    }
    }
```

نکته: زمانی می‌توان از کلیدواژه‌ی override استفاده کرد که قبلاً با همان نام، متدی در همان کلاس یا کلاس والد وجود داشته باشد.[۲، ۴]

۱۵.۲ نامریشن و استراکچر

کلاس‌ها تنها راه‌های تعریف انواع داده‌ای در زبان شئ‌گرای سوئیفت نیستند، نامریشن ها [۱۱] و استراکچرها [۱۲] کاربرد و قابلیت‌هایی مشابه کلاس را دارند، اما در شرایط متفاوتی به کار گرفته می‌شوند.
در سوئیفت، نامریشن یک نوع مشترک برای گروهی از مقادیر مرتبط تعریف کرده است و به برنامه‌نویس امکان می‌دهد در کد خود با این مقادیر به صورت type-safe کار کند. برای تعریف آن‌ها باید از کلیدواژه‌ی enum استفاده کرد. سوئیفت یک زبان است که جانب ایمنی نوع یا safety type را رعایت می کند؛ بدین معنی که اگر بخشی از کد برنامه انتظار مقداری از جنس رشته را داشته باشد، ویژگی type-safe مانع از این می‌شود که برنامه‌نویس به طور تصادفی مقداری از نوع عدد صحیح را به آن ارسال کند. برای مثال می‌توان به وضعیت اتصال شبکه اشاره کرد که در آن چهار حالتunknown connected connecting disconnected ممکن است. در این سناریو یک enum از نوع int تعریف می‌شود که مقدار خام آن از ۱ شروع می‌شود و به همین ترتیب ادامه می‌یابد.

```
enum Rank: Int {
    case Ace = 1
    case Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten
    case Jack, Queen, King
    func simpleDescription() -> String {
    switch self {
    case .Ace:
    return ”ace”
    case .Jack:
    return ”jack”
    case .Queen:
    return ”queen”
    case .King:
    return ”king”
    default:
    return String(self.rawValue)
    }
    }
    }
    let ace = Rank.Ace
```

[11] Enumeration
[12] Structure

```
let aceRawValue = ace.rawValue
```

در مثال بالا نوع مقدار خام (مقدار از نوع رشته، عدد صحیح، کاراکتر) تخصیص داده شده به ساختار نامریشن از جنس عدد صحیح است، بنابراین باید فقط مقدار اولین عضو را مشخص کرد و دیگر اعضای این ساختار به ترتیب مقداردهی می‌شوند (برای مثال اگر مقدار اولین عضو را ۱ بدهید، مقادیر دیگر اعضا به ترتیب ۲، ۳ و ۴ خواهند بود). جهت دسترسی به مقدار خام و اولیه‌ی اعضای ساختار ،enum کافی است از پروپرتی‌ای به نام rawValue استفاده کرد.

در سوئیفت،استراکچرها بسیاری از رفتارها و امکانات کلاس نظیر متد و مقداردهی را پشتیبانی می‌کنند. یکی از تفاوت‌های عمده‌ی استراکچر با کلاس در این است که به هنگام ارسال استراکچر به بخش‌های مختلف کد، یک کپی از آن به کد موردنظر فرستاده می‌شود. این در حالی است که کلاس با ارجاع به کد پاس داده می‌شود (بدین معنی که هرگونه تغییر در کلاس توسط کد، بر روی کلاس اصلی نیز اعمال می‌شود).
آن‌ها برای تعریف انواع داده‌ای سبک که به قابلیت‌هایی همچون ارث بری و تبدیل نوع نیاز ندارند، بسیار مناسب می‌باشند. جهت تعریف یک استراکچر جدید کافی است از کلیدواژه‌ی struct استفاده کرد:[۳]

```
struct Card {
    var rank: Rank
    var suit: Suit
    func simpleDescription() -> String {
    return "The suit.simpleDescription())"
    }
    }
    let threeOfSpades = Card(rank: .Three, suit: .Spades)
    let threeOfSpadesDescription = threeOfSpades.simpleDescription()
```

۱۶.۲ پروتکل

این الگو سپس برای پیاده‌سازی اعضای (متدها، پراپرتی‌ها و ...) یک enumeration ، کلاس یا استراکچر بکار گرفته می‌شود. هر یک از ساختارهای ذکر شده که الگو یا پروتکل را پیاده‌سازی کنند، در اصطلاح به آن conform (از آن الگو پیروی) می‌کنند. برای تعریف یک پروتکل، باید از کلیدواژه‌ی protocol استفاده کرد:

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    func adjust()
    }
```

نکته: دستور get ، پس از متغیر simpleDescription در کد بالا، بیانگر این است که مقدار متغیر نام برده، فقط خواندنی است؛ بدین معنی که مقدارش را می‌توان خواند اما امکان تغییر در آن وجود ندارد. پروتکل‌ها می‌توانند اینستنس‌متدها[۱۳] (متدهایی که به نمونه ی ایجاد شده از ساختار مورد نظر تعلق دارند و تمامی قابلیت های آن ساختار را پشتیبانی می‌کنند) و تایپ‌متدهایی [۱۴](متدهایی که در ساختار خاصی تعریف می‌شوند و به آن وابسته و مرتبط هستند) که توسط ساختار مورد نظر پیاده سازی می‌شوند را تعیین کنند. این متدها به عنوان بخشی از تعریف پروتکل و بدون بدنه مشخص می‌شوند.
کلاس‌ها، استراکچرها و نامریشن ها برای پیروی از پروتکل یا الگوی پیاده‌سازی خاص، اسم آن را پس از اسم خود و دو نقطه ذکر می‌کنند. هر ساختار می‌تواند از بی نهایت پروتکل پیروی کند. اسم پروتکل‌ها به صورت یک لیست تفکیک شده با ویرگول، پس از اسم ساختار مورد نظر عنوان می‌شوند. چنانچه یک کلاس از کلاس والدی ارث بری کند، در آن صورت اسم کلاس والد باید پیش از اسم پروتکل‌ها لیست شود.[۲، ۴]

[13] instance method
[14] type method

فصل ۳

مقایسه سوئیفت با زبان‌های دیگر

۱.۳ مقایسه با زبان پایتون

زبان سوئیفت بیشتر شبیه به پایتون است و هر دو یادگیری راحتی دارند. جایی که سوئیفت برتری خودش نسبت به پایتون را به نمایش می‌گذارد، مربوط به سرعت اجرا است. سوئیفت مانند کامپایلر LLVM به زبان ماشین کامپایل می‌شود، یعنی این که پایتون می‌تواند از چند نخی پشتیبانی کند، چیزی که پایتون هنوز در حال دست و پنجه نرم کردن با آن است.
در شرایطی که سرعت توسعه مهمتر از سرعت اجرا باشد، استفاده از پایتون می‌تواند بسیار مفید باشد. البته سوئیفت هم به تبعیت از پایتون چنین ویژگی را در خود جا داده است چرا که می‌تواند به کمک محیط توسعه یکپارچه ایکس‌کد، از حالت پلی‌گراند در سوئیفت برای افزایش سرعت توسعه بهره برد. پایتون دامنه وسیعی از موارد استفاده دارد، اما در درجه اول برای توسعه‌ی پایدار استفاده می‌شود.

با این حال از آن جا که زبان برنامه‌نویسی سوئیفت در مقایسه با پایتون نسبتاً جدید است، فعلاً نتوانسته است قدرت کافی برای به زیر کشیدن پایتون پیدا کند. پس در حال حاضر پایتون تمام مزایا و برتری‌ها را در اختیار دارد. همچنین سوئیفت دراصل به عنوان مکملی برای زنجیره ابزارهای اپل ایجاد شده است در حالی که پایتون چنین وابستگی‌ای نداشته و کاملاً مستقل است.[۶]

۲.۳ مقایسه با آبجکتیوسی

سوئیفت جدیدترین زبان برنامه‌نویسی شرکت اپل است، زبانی هم‌طراز با آبجکتیوسی است اما خواندن راحت‌تری دارد. اپل این زبان را به عنوان زبانی ساده‌نگر و با قدرتی چشم‌گیر و ۴ برابر شئ‌گراتر از زبان پیشین تلقی می‌کند. بزرگترین مزیت سوئیفت در یادگیری آسان آن است.
بزرگترین تفاوت کارکردی و البته بینشی سوئیفت و آبجکتیوسی در نگرش آن‌ها به مفهوم شئ‌گرایی OOP [1] است. OOP یک شیوه‌ی برنامه‌نویسی است که ساختار یا بلوک اصلی اجزای آن، شئ‌ها می‌باشند. اپل، آبجکتیوسی را بر پایه زبان سی و اضافه کردن مفهوم شئ‌گرایی به آن، تولید کرد و توسعه داد. در این زبان هدرها و ویژگی‌ها و توابع به گونه‌ای بیرون از فایل ایجاد شده بودند و با فراخوانی به فایل آبجکتیوسی اضافه می‌شدند اما در سوئیفت، فایل سوئیفت ایجاد شده، به خودی خود حاوی تمامی هدرها و ویژگی‌ها و کلاس‌های مورد نیاز است. این ساده‌سازی به توسعه‌دهنده کمک می‌کند که با سرعت، اطمینان و امنیت بیشتری به توسعه‌ای قائم بر شئ‌گرایی بیاندیشد.

[1] Object-Oriented programming

ـ شباهت‌ها: • انواع عددی مبنا. (Int,UInt,Float,Double)
• از کروشه‌ها برای آرایه‌ها استفاده می‌شود تا آن‌ها را اعلام کنند و یک مقدار از یک شاخص معین را از یکی از آن‌ها بگیرند.
• متدهای کلاس به ارث برده می‌شوند.
• قاعده شمارشی مشابه (for-in)

ـ تفاوت‌ها:
در زبان سوئیفت برخلاف آبجکتیوسی به صورت زیر است:
• عبارات نیازی نیست با نقطه ویرگول (;) به پایان برسند، اما برای نوشتن بیش از یک عبارت در یک خط باید از آن استفاده شود.
• عدم وجود فایل هدر
• استنباط کردن نوع
• برنامه‌نویسی رایج
• توابع، اشیای درجه یک هستند.
• عملگرها را می‌توان برای کلاس‌ها مجدداً تعریف کرد(سربارگذاری عملگرها) و عملگرهای جدیدی را می‌توان تعریف کرد.
• رشته‌ها به طور کامل از یونیکد پشتیبانی می‌کنند. بیشتر کاراکترهای یونیکد را می‌توان در شناسه‌ها یا عملگرها استفاده کرد.
• چند رفتار مستعد در برابر خطا در زبان‌های خانواده سی تغییر یافته‌اند.
• نیازی به استفاده از عبارت break در بلوک‌های سوییچ وجود ندارد.
• متغیرها و ثابت‌ها همیشه مقداردهی اولیه شده‌اند و آرایه‌ها همیشه بررسی شده‌اند.
• سرریزهای عدد صحیح که منجر به رفتارهای نامشخص برای اعداد صحیح امضا شده در زبان سی می‌شوند به‌عنوان یک خطای ران‌تایم در سوئیفت مشخص شده‌اند.
• شکل انفرادی if و while که اجازه حذف کروشه‌ها از عبارات را می‌دهد، پشتیبانی نشده است.
• عملگرهای کاهش و افزایش پس و پیش (i++, --i) ازسوئیفت سه به بعد پشتیبانی نشده‌اند، مخصوصاً زمانی که for به سبک زبان سی ازسوئیفت سه به بعد پشتیبانی نمی‌شوند.
• تساوی‌ها مقداری حاصل نمی‌کنند. این کار با دادن خطا در زمان کامپایل کردن، از خطای رایج نوشتن i=0 به جای i==0 جلوگیری می‌کند.[۲، ۷]

۳.۳ مقایسه با گو

به وجود آورنده زبان سوئیفت، اپل و برای زبان گو، گوگل [۲] است. سوئیفت بر روی یک وظیفه‌ی خاص متمرکز می‌شود،برنامه‌های آی‌اواس را طراحی می‌کند، در حالی که هدف گو برای یک کار بسیار گسترده و نوشتن سرورهای عمومی بک‌اِند [۳] و توسعه وب [۴] است. سوئیفت از تایپ ها رایج پشتیبانی می‌کند اما گو از آن‌ها پشتیبانی نمی‌کند و زبان سوئیفت ذاتاً پلی‌گراند دارد ولی گو فقط یک نمونه آنلاین از آن را دارد.

نحو سوئیفت بیشتر شبیه به زبان راست می‌باشد در حالی که نحو گو بیشتر به پایتون نزدیک است و یک فرم طولانی مانند سی‌پلاس‌پلاس است. نحو زبان گو، از تغییر روی زبان سی که کد را خوانا و فشرده می‌کند، آمده است. این عمل نحوهای لغوی اضافه می‌کند که تکرار راحت‌تر بر روی ساختار داده‌های مجموعه را اجازه می‌دهد. همه ویژگی‌های این زبان و ابزارهایش از الگوی یونیکس پیروی می‌کند و بیشتر روی منطق برنامه کار می‌کند.

نحو سوئیفت هنوز هم ممکن است تغییر پیدا کند درحالی که گو در طبیعت پایدار است. در زبان گو برای اجرای کد، نیازی به اجرا آن نیست و به صورت خودکار جمع‌آوری و اجرا می‌شود. قدرت اصلی زبان گو این است که حداقل و سریع است و به عنوان یک ابزار بسیار قدرتمند در هنگام برنامه‌نویسی وب، سرورهای میکرو و توسعه تلفن همراه عمل می‌کند. اجرا در گو سریع‌تر از سوئیفت می‌باشد. در بسیاری از موارد استفاده، ثابت شده توسعه وب در گو سریع‌تر از سوئیفت است. [۸]

۴.۳ مقایسه با کاتلین

سوئیفت و کاتلین دو زبان عالی برای آی‌اواس و اندروید [۵] هستند. هر دو دارای ویژگی‌های بسیار مدرن و نحو هستند که می‌تواند بشدت برای ساخت برنامه‌های محلی کمک کند.

سوئیفت، کلاس داده‌ای و کلاس‌های delegated یا ویژگی‌های delegated ندارد. سوئیفت، حاشیه‌نویسی را اجازه نمی‌دهد. در کاتلین، کلاس‌ها به طور پیش‌فرض نهایی هستند و ساختار و یا انتقال داده با مقدار ندارد. tuples و typealias و جمله حفاظتی ندارد. این دو زبان نحوهای متفاوتی دارند. ازجمله موارد جدول زیر:[۲]

[2] Google
[3] Backend
[4] Web
[5] Android

سوئیفت	کاتلین
func	fun
let	val
nil	null
init	constructor
protocol	trait

۵.۳ مقایسه با جاوا

زبان برنامه‌نویسی سوئیفت از قالب‌های متفاوتی استفاده می‌کند.و از آن برای پروژه‌های آی‌اواس استفاده می‌شود و نحو مختصری دارد و دارای امنیت و زبانی سریع است.
در زبان برنامه‌نویسی جاوا، قالب محاسباتی است که اولین بار توسط سان ماکروسافت در سال ۱۹۹۵ منتشر شد. برنامه‌های زیادی وجود دارد و وب سایت‌هایی که فقط با نصب کردن جاوا کار می‌کنند. جاوا سریع، امن و قابل اطمینان است. سوئیفت برای توسعه‌ی نرم‌افزار سیستم‌عامل اپل استفاده می‌شود. از سوی دیگر جاوا جهانی است. این می‌تواند برای نوشتن برنامه‌ها برای وب، سرورها، سیستم‌عامل ویندوز و لینوکس و مک و اندروید استفاده شود. با این وجود یادگیری هر دو زبان بهتر است اما سوئیفت یک مزیت برتری نسبت به جاوا دارد و آن خاصیت زمین بازی می‌باشد.[۲]

فصل ۴

نتیجه گیری

این پژوهش به منظور«آشنایی با زبان سوئیفت، ویژگی ها، مقایسه با دیگر زبان‌های پرکاربرد و آموزش مختصر این زبان» انجام شده است.

براساس یافته‌ها، برنامه‌نویسی زبان سوئیفت دارای ویژگی‌های بسیار کارآمدی از جمله یادگیری آسان، سرعت بالا، امنیت و ... می‌باشد. بسیاری از برنامه‌ها با زبان سوئیفت نوشته می‌شوند و امروزه بازار داغی دارند، پس یادگیری این زبان برنامه‌نویسی تأثیر چشم‌گیری در پیشرفت در این زمینه دارد.

همان‌طور که گفته شد زبان سوئیفت هم‌طراز با زبان آبجکتیوسی است اما سادگی بیشتری دارد به همین دلیل در شرکت اپل از زبان برنامه‌نویسی سوئیفت استفاده می‌شود. زبان سوئیفت در مقایسه با گو و پایتون فرم کوتاه‌تری دارد. زبان سوئیفت بیشتر شبیه به پایتون است و هر دو یادگیری آسان دارند اما سرعت اجرا در سوئیفت بیشتر می‌باشد اما در صورتی که سرعت توسعه نسبت به سرعت اجرا اولویت داشته باشد، پایتون ترجیح داده می‌شود. با این وجود که سوئیفت زبان نسبتاً جدیدی است و هنوز نتوانسته است جایگزین پایتون شود اما این حدس وجود دارد که در آینده نزدیک قدرت آن چندین برابر بیشتر خواهد شد. سوئیفت و کاتلین هر دو زبان عالی برای آی‌اواس و اندروید هستند و ویژگی‌ها و نحوهای مدرن و متفاوتی دارند. سوئیفت و جاوا هر دو نحو مختصری دارند و دارای امنیت و زبانی سریع هستند و با این وجود یادگیری هر دو زبان خوب است اما سوئیفت یک مزیت برتری نسبت به جاوا دارد و آن خاصیت زمین بازی می‌باشد پس سوئیفت نسبت به آن برتری دارد.

واژه‌نامه فارسی به انگلیسی

سوئیفت ..... Swift
پایتون ..... Python
آبجکتیوسی ..... Objective-C
گو ..... Go
کاتلین ..... Kotlin
جاوا ..... Java
واچ‌اواس ..... watchOS
مک‌اواس ..... macOS
آی‌اواس ..... iOS
تی‌وی‌اواس ..... tvOS
چارچوب ..... framwork
کوکا ..... Cocoa
کوکا تاچ ..... Cocoa Touch
ایکس‌کد ..... Xcode
لینوکس ..... Linux
زمان اجرا ..... runtime
کریس لانتر ..... Chris Lanther
سی‌ال‌یو ..... CLU
سی C
رابی ..... Ruby
راست ..... Rust
کنفرانس جهانی توسعه‌دهندگان اپل ..... WWDC
نحو ..... syntax
مکینتاش ..... Macintosh
وی ام ور ..... vmware
ویرچوآل‌باکس ..... virtualbox
اسمال‌تاک ..... Smalltalk
نماد نقطه ..... dot notation

فضای نام ..... Namespace
گسترش قراردادی ..... protocol extensibility
متن باز ..... Open Source
قالب ..... Platform
زمین بازی ..... Playground
برنامه ..... Application
بسته ..... closure
لیست ضبط ..... Capture List
هیپ ..... Heap
نابودی ..... dereferencing
استراکت ..... Struct
نمونه ..... Instance
اکسسور ..... Accessor
واحد مدیریت حافظه ..... MMU
متغیر ..... Variable
اختیاری ..... Optional
ثابت ..... Constant
آرایه ..... Array
توضیحات ..... comments
دیکشنری ..... Dictionary
پیش‌فرض ..... defult
پراپرتی ..... Property
برنامه نویسی شیء گرایی ..... Object-Oriented Programming
ویژگی‌ها ..... Properties
گوگل ..... Google
بک‌اِند ..... Backend
وب ..... Web
اندروید ..... Android
نامریشن ..... Enumeration
استراکچر ..... Structure
سوییچ ..... Switch
پروتکل ..... Protocol

کلمات کلیدی فارسی

کلمات کلیدی

- **سوئیفت**
- **امنیت**
- **توسعه سریع**
- **متن‌باز**
- **زمین بازی**
- **بازارکار**
- **مدیریت حافظه**
- **برنامه‌نویسی**
- **مقایسه با پایتون**
- **مقایسه با آبجکتیوسی**
- **مقایسه با گو**
- **مقایسه با کاتلین**
- **مقایسه با جاوا**

کلمات کلیدی انگلیسی

keywords

- **Swift**
- **Security**
- **Fast development**
- **Open Source**
- **Playground**
- **Labor market**
- **Memory Management**
- **Programming**
- **Compare with Python**
- **Compare with Objective-C**
- **Compare with Go**
- **Compare with Kotlin**
- **Compare with Java**

کتاب‌نامه

[1] https://en.wikipedia.org/wiki/Swift_(programming_language). .

[2] Jon Manning , Paris Buttfield-Addison , Tim Nugent . Learning Swift (Building Apps for MacOs , IOS , and Beyond).

[3] https://www.udacity.com/course/learn-swift-programming-syntax--ud902/ . .

[4] http://xcode.blog.ir/post/4/learning-swift-farsiXcode/. .

[5] https://aryagostarafzar.com/shop/swift/. .

[6] https://www.cleveroad.com/blog/python-vs-swift/. .

[7] https://medium.com/swiftify/swift-vs-objective-c-comparison-32aba9dad4e3.

[8] https://www.educba.com/swift-vs-go/ .

[9] https://www.zoomit.ir/2017/9/27/212674/12-reasons-to-learn-apples-open-source-swift-language/. .

Abstract

This article intends to introduce Swift programming language. Language features (security, rapid development, open source, playground, etc.) are expressed in Swift, and the demand for it and its market, and how it is managed, is discussed, and it speaks with other widely used programming languages, including, Python , Objective-C, Go, Kotlin and Java, to show its growth and popularity, and helps the developer select the programming language in his / her own choice and obtain a good and accurate program.

Briefly Swift is taught. This language is very extensive and requires more resources than one article to learn. But in this article, all efforts have been made to make important applications.

College of Science
School of Mathematics, Statistics, and Computer Science

Familiar with programming language Swift

Asma Arab

Supervisor: Engineer Ebrahim Nagibzadeh Mashayekh

A thesis submitted to Graduate Studies Office
in partial fulfillment of the requirements for the degree of
B.Sc in
Computer Science

2019