Programming Language Reference

	Python	TypeScript	Go	Scala	Haskell
paradigms	OO, imperative, functional	OO, imperative, functional, generic	imperative, functional	functional/OO hybrid	pure functional
typing discipline	dynamic, duck, goose, gradual	dynamic, structural, duck, gradual	static, strong, structural	static, inferred, nominal with structural features	static, inferred
install	pipenv https://docs.pipenv.org/	npm install -g typescript		brew install sbt http://docs.scala-lang.org/getting-started.html	brew install stack https://docs.haskellstack.org/en/stable/README/
docs	https://docs.python.org			https://www.scala-lang.org/api/current/	https://www.haskell.org/hoogle/
hyperpolyglot	http://hyperpolyglot.org/scripting	https://hyperpolyglot.org/web		http://hyperpolyglot.org/rust	http://hyperpolyglot.org/ml
Libraries	https://github.com/vinta/awesome-python			https://github.com/lauris/awesome-scala	
build/env tooling	conda or pipenv invoke	npm		sbt	stack
new project	conda create --name snakes python=3.8 conda activate <project_name> conda install <package_name> conda env list	tsc --init npm init		sbt new scala/scala-seed.g8	stack new my-project cd my-project stack setup stack build stack exec my-project-exe

repl	$ python
	>>> exit()
	>>> <ctrl-d>
		https://github.com/TypeStrong/ts-node
		$ ts-node
				$ sbt console
				$ sbt consoleQuick
					$ stack ghci
					$ stack repl
					Prelude> ctrl-d

jupyter	default	tslab https://github.com/yunabe/tslab		Almond https://almond.sh/	IHaskell https://github.com/gibiansky/IHaskell
online repl	https://www.python.org/shell/	https://www.typescriptlang.org/play	https://play.golang.org/	https://replit.com/languages/scala	https://replit.com/languages/haskell
repl commands	jupyter x? — introspection x?? — source code %run file.py %load file.py %cpaste <paste> --	> .type x		> :load foo.scala > :t foo _	-- load file Prelude> :l foo.hs -- reload file Foo> :r -- type Foo> :t map -- kind Foo> :k Num
multiline repl				> :paste	> :{ ... :}
interpreter	$ python foo.py	$ ts-node foo.ts		$ scala Foo.scala	n/a
cmdline program	python -c 'print("foo")'			scala -cp . -e 'pkg.Foo.f'	n/a
compile to executable	n/a			$ scalac Foo.scala $ scala -cp . Foo	$ ghc foo.hs $ ./foo
define module	module per .py file			package foo.bar (in /foo/bar/_.scala)	module Foo.Bar where
use module	from collections import deque import collections			import foo.Bar import foo.{Bar, Baz} import foo._	import Foo.Bar -- all functions in Bar now in scope without qualification
default import	import os, re, sys			import java.lang._ import scala._ import scala.Predef._	Prelude
IDE	PyCharm	VSCode		intellij: Add Scala Plugin, Add Scala SDK: /usr/local/Cellar/scala/2.11.8/idea/lib	https://wiki.haskell.org/IDEs
naming conventions	snake_case	snake_case		lowerCamelCase	snake_case

Syntax & Semantics

	Python	TypeScript	Go	Scala	Haskell
Basic Syntax					

main	def main():
	    println("a")
	
	if __name__ == '__main__':
	    main()
			package main
			
			func main() {}
				object Hello extends App {
				    println("a")
				}
				
				object Hello {
				  def main(args: Array[String])
				{
				    println("a")
				  }
				}
					main = do
					        putStrLn("a")

sequenced computation	by default	by default		by default, but idiomatic with for comprehension:
				for {
				  a <- f1()
				  b <- f2(a)
				  c <- f3(b)
				} yield {
				  c
				}
					do
					  expr1
					  expr2

EOL comment	#	//		//	-- comment
multiline comment	n/a	/* */		/* */	{- comment another comment -}

line sep / statement terminator	newline or ;
	
	newlines not separators inside (), [], {}, triple quote literals, or after backslash: \
		; or newline
		
		newline not separator inside (), [], {}, "", '', or after binary operator
		
		newline sometimes not separator when following line would not parse as a valid statement
				; or sometimes newline
				
				A newline does not terminate a statement:
				  (1) inside ( ) or [ ],
				  (2) if the preceding line is not a complete statement,
				  (3) if following token not legal at start of a statement.
					next line has equal or less indentation, or ;

scoped block	n/a			{ }	offside rule or { } f x = x + r where r = 42
types	(optional) None bytes int float bool str list dict set frozenset array int float complex fractions rationals decimal	(optional) * string * boolean * number * symbol * any * unknown * never * void * bigint	bool string int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 uintptr byte // alias for uint8 rune // alias for int32 // represents a Unicode code point float32 float64 complex64 complex128	Int Integer Long Float Double BigDecimal BigInt Fractional	Bool Int Int32 Word Integer Float Double Rational Fixed Ex Scientific
immutable variable	n/a	const x: number = 1 const x = 1	const i int = 1	val x = 1	x = 1 let x = 1
mutable variable	x = 1	let x = 1	var i int = 1 i := 3	var x = 1	lol
lazy eval variable				lazy val x = 1	
print a string	print("a") print("a", end="")			print("a") println("a")	print "a" putStrLn "a" putStrLn . show $ "a"
explicit type definition	x: int = 1	const x: number = 1		1 : Int val x: Int = 1	1 :: Int
boolean	True False	true false		true false	data Bool = True | False
falsey	False, None, 0, 0.0, 0j, Decimal(0), Fraction (0,1), '', (), [], {}, set(), range(0)	false null undefined "" 0 NaN		false	False
logic	not or and	! || &&		! || &&	not || &&
null	None	null	nil	null scala.Null None	n/a
symbol		const symbol1 = Symbol("x");			
null test	x is None	x === null		x == null	n/a
value comparators	== != < > <= >=	=== !== < > >= <= with type coercion: == !=		== != < > <= >=	== /= > < >= <=
identity comparators	is is not				
unknown		unknown			
		typeof maybe === "string" x instanceof Foo			
bottom	Any			scala.Nothing	undefined
top/any	Any	any		Any	
void	None	void		() — Unit	
never		never			
type assertions		x as string <string>x			
literal types		let name: 'Foo Bar'; // type is 'Foo Bar'			
type guard		"attr1" in x (o: any): o is Foo => !!o && "attr1"			
tertiary	True if True else False				

if	if p:
	  e1
	else:
	  e2
			if x < 0 {
			  return "a"
			} else {
			  return "b"
			}
			
			if x := 1; x < lim {
			  return x
			}
				if (p) then e1 else e2
				if (p) {
				  e1
				} else {
				  e2
				}
					if c then e1 else e2

math package	import math https://docs.python.org/3/library/math.html			import scala.math._ https://www.scala-lang.org/api/current/scala/math/index.html	
arithmetic	+ - * / // %			+ - * / %	+ - * / div mod quot rem
Functions / Methods					

function definition	def f(x): return x
	
	def f(a, *bs, c_kw_only, **d)
	
	def f(a, *, b_kw_only)
	
	def tag(a, /, b)
		function f(x: number) : number
		{
		 return x
		}
			func f(x int) string {
			  ""
			}
				def f(x: Int):Int = x
				val f = (i: Int) => i + 1
				val f: Int => Int = (i) => i + 1
				val f = (i: Int) => { i + 1 }
				val f: Int => Int = (i) => { i + 1 }
				def f[A](a: A): String = a.toString
					f :: Int -> Int
					f x = x
					let f x = x

function invocation	f() f(1) f(1, 2, 3) f(p1=1, p2=2, p3=3)	f(1)		f f() f(1) f(1, 2, 3) f(x=1)	f 1
variadic parameter definition	def first_and_last(*args, **kwargs) # args is an array # kwargs is a map			def f(xs: String*)	
variadic parameter invocation	f(*xs)			f(xs:_*)	
function placeholder	def f: pass			def f = ???	f = undefined
lambda / anonymous function	lambda x, y: x + y			x => x + 1 (x, y) => x + y _ + _	\x -> x + 1 \x y -> x + y \x y -> +
curryable function				def f(a: Int)(b: Int): Int	by default
curry a function				f_2args.curried(1)	f_2args 1 (*) 1
partial application	functools.partial(f, arg)			f_2args(1, _:Int)	n/a
function composition				f compose g g andThen f	f . g
function application				f.apply(1)	f $ 1
example function composition					putStrLn . show . take 3 . reverse $ "Hello"
infix in prefix				n/a	(*) 42 42
prefix in infix				n/a	42 \`f\` 42
right associative				starts with : e.g., 1 :: Nil	
infix/ method				a + b a.+(b) a.foo(b) a foo b	
tail recursive				last action is tail call, with @tailrec for validation	default
C-style for			for i := 0; i < 10; i++ { }		

comprehension	[f(x) for x in xs if p] #listcomp
	(f(x) for x in xs if p) #genexp
				for {
				  x <- xs
				  y <- ys
				  if x + y == 0
				} yield (x, y)
					[a | a <- xs, a <= n]

pattern matching or switch		switch (x) {
		    case "a": return 1;
		    case "b": return 2;
		}
			switch x := "a"; x {
			      case "a": 1
			      case "b": 2
			      default: 3
			}
			
			switch {
			      case true: 1
			      default: 2
			}
				x match {
				  case Nil => "empty list"
				  case x::xs => "non-empty list"
				  case 0 => "zero"
				  case 1 | 3 => "odd"
				  case x if x.foo > 42 "y"
				  case b : Bar => b
				  case foo @ Foo(_, bar @ Bar(v)) => (foo, v)
				  case _ => "hole!" // can't use _
				}
					f t@(a, _) = do
					  print a
					  return t

defer			defer		

function parameter destructure	n/a			n/a, but can do case blocks
				
				val f: (List) = {
				  case Nil => true
				  case _ => false
				}
					f :: [Char] -> [Char]
					f [] = []
					f x:xs = xs
					f x = [x]

try/catch	try:
	  f()
	except ValueError:
	  handle_exception()
	else:
	  no_exception_occured()
	finally:
	  regardless_now_do()
				Try { x() }
				Try = Success(x) | Failure(e)
				
				try {
				  Some(x())
				 } catch {
				    case ioe: IOException =>
				         logger.error(ioe)
				         None
				    case fnf: FileNotFoundException =>
				         logger.error(fnf)
				         None
				  }	

non-strict arguments				f(a: => T){ // thunk
				  lazy val aEvaled = a
				}	

Data Structures					
array	array	string[] Foo[]		Seq Array Vector	
list	list			List HList (shapeless)	list
List-like data structures					
	List	Array		List, Seq, Array, Vector	List
type	list[T]	T[]		List[T]	List[T]
empty ds	[] -- technically an array	[]		Nil List() Seq()	[]
is empty?	not []			xs == Nil xs.isEmpty xs.nonEmpty	null xs
construction	[1, 2, 3] list(1, 2, 3)	[1, 2, 3]		Nil List(1,2,3) 1 :: 2 :: 3 :: Nil Seq(1, 2, 3) Seq()	[1, 2, 3, 4]
slice	s[a:b:c]				
first / car	xs[0]			xs.head xs.headOption	head xs
rest / cdr	xs[1:]			xs.tail	tail xs
last	xs[-1:]			xs.last xs.lastOption	last xs
all but last	xs[:-1]			xs.init	init xs
length/ size	len(xs)			xs.length	length xs
nth	xs[n]			xs(n)	xs !! 1
min/max	min(xs) max(xs)			xs.min xs.max	minimum xs maximum xs
take	xs[0:n]			xs take n	take 1 xs
drop	xs[n:]			xs drop n	drop 1 xs
concat	xs + xs2			xs ++ xs2 List.concat(xs, xs2) xs ::: xs2	xs ++ xs2 concat xs xs2
reverse list	list(reversed(xs))			xs.reverse	reverse xs
xs w/ x in position n	xs[n] = x			xs updated (n, x)	let (ys,zs) = splitAt n xs in ys ++ [new_element] ++ (tail xs)
cons (add to head)				4 :: xs	42 : xs
add to tail				xs :+ x xs ::: List(x)	xs ++ [x]
add to middle				val (ys, zs) = xs splitAt n ys ::: new_element :: zs	let (ys,zs) = splitAt n xs in ys ++ [new_element] ++ zs
				Seq ++ concat two collections, type left ++: concat two collections, type right +: append x +: xs (general ::) :+ append xs :+ x /: foldl (0 /: xs)(_+_) :\ foldr (xs :\ 0)(_+_)	
	https://docs.python.org/3/library/functools.html https://docs.python.org/3/library/itertools.html				

functions	operator
	      Math operations: add(), sub(), mul(), floordiv(), abs(), …
	
	      Logical operations: not_(), truth().
	
	      Bitwise operations: and_(), or_(), invert().
	
	      Comparisons: eq(), ne(), lt(), le(), gt(), and ge().
	
	      Object identity: is_(), is_not().				

exists	any(xs)			xs exists p	
forall	all(xs)			xs forall p	all xs p
contains				xs contains x	any xs p elem x xs
index of				xs indexOf x	
take while	itertools.takewhile(predicate, iter)			xs takeWhile p	
drop while	itertools.dropwhile(predicate, iter)			xs dropWhile p	
span				xs span p	
sort with				xs sortWith p	
sort	sorted(xs) xs.sort (mutation)			xs.sorted	
group by	itertools.groupby(iter, key_func=None)			xs groupBy f	
map	map(f, iter1, iter2, …)			xs map f	
flatmap				xs flatMap f	
fold left	itertools.accumulate(iterable[, func, *, initial=None])			xs.foldLeft(init)(f) def foldLeft[B](z: B)(op: (B, A) => B): B	foldl f xs
fold right				xs.foldRight(init)(f) def foldRight[B](z: B)(op: (A, B) => B): B	foldr f xs
fold (arbitrary)				xs.fold(init)(f) def fold[A1 >: A](z: A1)(op: (A1, A1) => A1): A1	
scan				scanRight scanLeft	
reduce	functools.reduce(function, iterable[, initializer]) # also foldl			reduce[B >: A](op: (B, B) => B): B xs reduce f xs reduceLeft f xs reduceRight f xs reduce*Option f	
filter	filter(p, xs)			xs filter p	
filter not	itertools.filterfalse(predicate, iter)			xs filterNot p	
partition				xs partition p	
flatten				xss.flatten	
zip	zip(iterA, iterB, …)			xs zip ys	
unzip				xs.unzip	
sum	sum(iter, start=0)			xs.sum	sum xs
product				xs.product	product xs
	sorted(iterable, key=None, reverse=False)				
	enumerate(iter, start=0)				
	any(iter) and all(iter)				
chain	itertools.chain(iterA, iterB, …)				
combinations	itertools.combinations(iterable, r)				
permutations	itertools.permutations(iterable, r=None)				
	iter(x, stop)				
Other Basic Types					
enum		enum Color { R, G, B, } enum ABCs { A = 1, B, C, } ABCs[2] enum Foo { Up = "UP", Down = "DOWN", } const enum — irreversable		sealed trait Color case object R extends Color case object G extends Color case object B extends Color	
tuple	(42, 'foo', True)	let x: [string, number] = ["hello", 10]		(42, "foo", true) nested pattern matching!	(1, "foo", True) (,) 1 "foo" True , is an infix operator!
tuple operators	t[0] # unpacking x, y, *rest = (1, 2, 3, 4, 5)	x[0]		t._1 t._2	fst t snd t swap t
vector / array	array.array			Vector(1, 2, 3) x +: xs // front xs :+ x // end	

	Python	TypeScript	Go	Scala	Haskell
Basic Syntax					
set	set() or frozenset(x) { x } setcomp {x for x in xs} S ∩ Z: s & z S ∪ Z: s | z S \ Z: s - z S Δ Z: s ^ z e ∈ S: e in s S ⊆ Z: s <= z s.intersection(it, …) s.union(it, …) s.clear() s.discard(e) s.remove(e) #KeyError if not			Set(1, 2, 3) S ∩ Z: s & z S ∪ Z: s | z S \ Z: s &~ z or s -- z S Δ Z: n/a e ∈ S: s.contains(e) S ⊆ Z: s.subsetOf(z) diff: &~ + add element ++ add another set - remove -- remove set	
deque	from collections import deque deque(range(10), maxlen=10) rotate, appendLeft, extend, extendLeft				

map/ dictionary	{'x': 1, 'y': 2}
	iteration over keys
	d.clear()
	k in d
	del d[k]
	d.get(k, [default])
	d[k] # error if not contains
	d.items()
	d.keys()
	d.values()
	d.pop(k, [default])
	d[k]
	d.update(m, [**kargs])
	d.setdefault(k, [default])
	
	dictcomp {x : v(x) for x in xs}
				Map("a" -> 1, "b" -> 2)
				map("c") // NSEE
				map get "c" // None
				Seq(Some("a"-> 1), Some("b"-> 2), None).flatten.toMap
				(Seq("a" -> 1) ++ opt.map { x => "b" -> x }).toMap	

map with default value	collections.defaultdict(list)			Map().withDefaultValue("foo")	
add to mutable map	d['a'] = 1			m + ("a" -> 1) m + ("a", 1) m ++ Map("a" -> 1)	
range	range(start, end, step) itertools.count(start, step) # lazy			1 until 10 1 to 10 by 2 List.range(start, endExclusive)	
tabulate				List.tabulate(5)(n => n * n)	
collect				collect(pf) collectFirst(pf)	
lazy collections				xs.view … to(Seq) (2.13) LazyList Stream (older)	default
Strings					
char					'a'
literal string	'foo' "foo" '''foo''' """foo"""	'foo' "foo"		"foo" """multiline foo"""	"foo"
interpolation	f'{x} {y}'	`{x} {y}`		s"$foo ${bar.f} ${floatVal}%.2f" s""""in quotes"""" f"format string" TBD	
raw	r'a literal backslash n \n'			raw"a literal backslash n \n"	
string conc.				"a" + "b"	"a" ++ "b"
bytes	s.encode('utf-8') bs.decode('utf-8')				
format string	'{0:.2f} {1:s} ${2:d}'.format(3.14159, 'x', 1)			"foo %s %d %.2f".format("bar", 7, 3.1415)	
casing	s.upper() s.lower()			"foo".toUpperCase "FOO".toLowerCase	
trim	s.strip() s.lstrip() s.rstrip()			" foo ".trim	
type conversion	int(s)			x.toString toInt toFloat	
join	','.join(xs)			xs.mkString(",")	
split	s.split(',')			"a b c".split(" ")	
partition	'a b c'.partition(' ') 'a b c'.rpartition(' ')				
length	len(s)			"foo".length	
substring	s[start:end]			"foo".substring(0, 1)	

regex	import re
	
	re.search("foo.*", text)
	
	m = re.match(pattern, string)
	m.group(0)
	
	r = re.compile(r'\w+')
	r.findall(s) / finditer
	r.match(s)
	r.fullmatch(s)
	r.split(s, maxsplit=1)
	r.sub(repl, s, count=1)
	re.escape(pattern)
	groupdict
	
	https://docs.python.org/3/library/re.html
				raw"[0-9]+".r.findFirstIn(s)
				findAllIn
				findFirstMatchIn
				replaceAllIn
				
				"2004-01-20" match {
				case raw"(\d{4})-(\d{2})-(\d{2})".r(year, month, day) => s"$year"
				}
				
				https://www.scala-lang.org/api/current/scala/util/matching/Regex.html	

F					
option/ maybe	from typing import Optional			Option = None | Some(x) getOrElse("<not assigned>") map match { case None … case Some(x) }	data Maybe = Nothing | Just(x)

option example				def toInt(s: String): Option[Int] = {
				try {
				Some(s.toInt)
				} catch {
				case e: Exception => None
				}
				}
				
				toInt(aString) match {
				case Some(i) => println(i)
				case None => println("Error: Could not convert String to Int.")
				}	

try				Try = Success(v) | Failure(e)	
either				Either = Left(a) | Right(b) right-biased	
future				Future.successful(v) Future.failed(v)	
User-defined Types					
union		let x: string | number;			
type alias		type Name = string;		type Name = String	type Name = String

ADT / named tuple / struct	collections.namedtuple
	typing.NamedTuple
	@dataclasses.dataclass
		type Foo = {
		attr1: string;
		optAttr1?: string;
		}
		
		let x: Foo = { attr1: "x" }
			type Foo struct {
			X int
			Y int
			}
			Foo{1, 2}
				case class Foo(attr1: String)	

class	class C2(C1) {
	#todo
	}
		class Foo extends Bar implements Baz, Qux {
		attr1: string;
		
		constructor(public readonly attr2: string){};
		}			

attribute visibility modifiers		public protected private		(none – public) protected protected[package] private	
abstract class	from abc import ABC class Foo(ABC): pass	abstract class Foo {}		abstract class Foo() { }	

trait/ interface		interface Foo {
		attr1: string;
		}			

ADT product type	from collections import namedtuple Foo = namedtuple('Foo', 'x y') or import typing Foo = typing.NamedTuple('Foo', [('x', str), ('y', float)]) or Foo = typing.NamedTuple('Foo', x=str, y=float) or from dataclasses import dataclass @dataclass(frozen=True)			sealed trait Foo final case class Bar extends Foo final case class Baz(c: Int, d: String) extends Foo	data Foo = Foo String Int data Foo = Foo { a :: String, b :: Int }
ADT sum type tagged union				sealed trait Foo case object Bar extends Foo case object Baz extends Foo	data Bool = False | True
ADT product of sums type					data Foo = Bar | Baz Corge deriving Grault
traits/ mixins				trait Foo { }	
single unary data constructor				@newtype case class C(v: String)	newtype
nullary data constructor				case object Foo	data Foo = Foo
unary data constructor				case class Foo(attr1: String)	data Foo a = Foo a

type parameters		interface Foo<T> {
		f(value: T): T;
		}
		
		function f<T> (x: T): T {
		return x;
		}
		
		const printMe = <T> (x: T): T => {
		return x;
		}			

I/O					

read file	with open(path, 'r') as f:
	lines = [x.rstrip() for x in f]
	or lines.readlines()
				val lines = Source.fromFile(filename).getLines.(toList|toArray|mkString)
				
				val bufferedSource = Source.fromFile("example.txt")
				for (line <- bufferedSource.getLines) {
				println(line.toUpperCase)
				}
				
				bufferedSource.close
				
				betterfiles	

write file	with open(path, 'w') as f: w.writelines(xs) w.write(s)				
read from stdin				scala.io.StdIn.readLine()	
JSON					
library	built-in			circe spray-json	
	import json json.dumps(x) pretty print – json.dumps(x, sort_keys=True, indent=4) json.loads('') https://docs.python.org/3/library/json.html				
optics				monocle	
Tests					
Other					
base64	import base64 base64.b64decode(str)				
random	import random random.randint(0, 99) random.random() random.gauss(0, 1)			val r = scala.util.Random r.nextInt(excl) r.nextDouble r.nextFloat Random.nextString(10) Random.alphanumeric.take(10).mkString	
Libraries					
Files				better-files https://github.com/pathikrit/better-files	
dates and times	from datetime import datetime, date, time dt = datetime(2021, 1, 20, 12, 30, 21) dt.strftime('%m/%d/%Y %H:%M') datetime.strptime('20091031', '%Y%m%d') datetime.timedelta(17, 7179) datetime.fromisoformat('2017-01-01T12:30:59.000000') datetime.isoformat() <= 3.6 python-dateutil				
arg parsing	https://github.com/docopt/docopt			https://github.com/scopt/scopt	
HTTP				akka-http http4s	
Functional				Cats scalaz	