Python programming language: introduction

Basic elements

Python program (script): sequence of statements, instructions or commands
that are interpreted and executed by the Python interpreter
using basic elements as values, types, variables, ...

Value: the core elements that a Python program manipulates

Type: range of values and operations that we can do with objects of this type.
Any value falls into a class or type

Scalar types: its values are indivisible
Non-scalar types: values have an internal structure.

Python: 3 three scalar basic types:

`int`	integer
`float`	real
`bool`	Boolean

```
>>> type (3)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>> type(True)
<class 'bool'>
```

Type `int`

Represents an interval of integers [MIN, MAX] 4 bytes int: $[-2^{31}, 2^{31}-1]$

values: sequence of digits and optional character – (negative) `3, -24, 6789, 0`

operators: `-` (opposite, **unary** operation)
`+, -, *, **, / , //, %` (**binary** operations)

result: `int`

```
>>> 3 + 4   # addition
7
>>> 3 - 4   # subtraction
-1
>>> 3 * 4   # product
12
>>> 2**3    # exponentiation
8
```

standard division result: `float`

```
>>> 6/4     # int divided by int
1.5
>>> 6/4.0   # int divided by float
1.5
>>> 4/2     # int divided by int
2.0
```

Type `int`: integer division

floor division (`//`) and modulo (`%`) operators

d (dividend) = q (quocient) * s (divisor) + r (remainder)

```
q = d // s
r = d % s
```

← "d modulo s" or "d mod s" for short


```
>>> 6//4
1
>>> 6 % 4
2
>>> 13//3
4
>>> 13 % 3
1
```

negative operands

```
>>> 7//2
3
>>> 7%2
1
>>> 7// -2
-4
>>> 7%-2
-1
```

Type `float`

Represents a subset of real numbers: limitation in range (magnitude) and precision
`float`: floating point representation

values: sequence of digits, optional – (for negative values)
and decimal point (dot character, mandatory)
`3.0, 3., -24.78, 6789.876, 0.0` in scientific notation: `2.3e-5` ($2.3 \cdot 10^{-5}$)

operators: `-` (opposite, **unary** operation)
`+, -, *, **, /` (**binary** operations)

hybrid `int/float` operations: +, -, *, **

operator	`int`	`float`
`int`	`int`	`float`
`float`	`float`	`float`

Remember:
division operation, /,
always gives a float result

```
>>> 3+4.4
7.4
>>> 16.44 * 0.5
8.22
>>> 16.44 /2
8.22
>>> 1.1 - 1.1
0.0
>>> 15/2
7.5
```

Type `bool`

Represents Boolean (logic) values: `False, True`

comparison operators: == (equals) != (not equal to),
operands: same type < (less than) <= (less than or equal to),
result: `bool` > (greater than) >= (greater than or equal to)

```
>>> 3 > 4
False
 >>> 2.5 < 3
True
```

Type `bool`

Boolean operators: `not`, `and`, `or`
Truth tables:

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

a	not a
True	False
False	True

chained comparisons allowed →

```
>>> 1 <= 6 <= 10
True
>>> 1 <= 6 and 6 <= 10
True
```

$6 \in [1, 10]$

Expressions

Combination of variables, values, operators and calls to functions, following syntactic rules
The Python interpreter **evaluates** an expression and gives a **result**

line format

mathematics	Python
2^4	`2**4`
$\frac{2+5}{3}$	`(2+5)/3`
$\frac{6}{1+2}$	`6/(1+2)`
$\frac{2x}{8}$	`2*x/8`
$\frac{8}{2x}$	`8/(2*x)`

rules of precedence

`**`	right-associative
`-`	opposite operator
`*,/,//,%`	left-associative
`+,-`	left-associative
`==,!=,...`	
`not`	
`and`	
`or`	

```
>>> 2**3**2
512
>>> 2**(3**2)
512
>>> (2**3)**2
64
>>> (2+3)*(4+5)
45
>>> 2+3*4+5
19
>>> 3+4<5  # (3+4)<5
False
```

Variables

A **variable** is a **name** or **identifier** that refers to a value

Variable names:

- sequence of characters: letters, digits and _ (underscore)
- initial character: letter or _
- case sensitive
- reserved words (keywords) forbidden

Examples:

Correct names: `area`, `Area`, `b`, `abc`, `x23`, `cercle_area`

Different variables

Compound name: use underscore, space is not allowed

Wrong names:

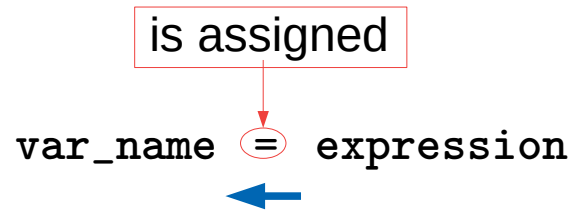
`1abc` begins with a digit

`cercle area` **space character not allowed**

`radius*` any other spacial character, as *, is not allowed

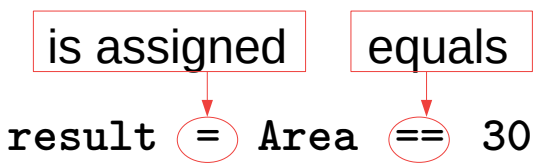
`float` keywords are not allowed

Assignment statement



- «*var_name is assigned expression*»
- assignment operator (=) is **not** commutative
- behavior:
  1) right-hand side expression is evaluated → value
  2) left-hand side variable name is assigned this value

```
>>> area = 5
>>> Area = area * 2
>>> area, Area
(5, 10)
>>> Area = Area + 10.5
>>> result = Area == 30
>>> area, Area, result
(5, 20.5, False)
>>> type(area)
<class 'int'>
>>> type(Area)
<class 'float'>
>>> type(result)
<class 'bool'>
```



Strings: `str`

Represents string characters:
character sequences as names, codes, phone numbers, ...

non-scalar, sequence type

values: quoted (single or double)
'abc-475', 'Tomorrow', 'aa@aaa.aa' , "Hanna's bag"

operators (block): + (concatenation), * (replication)
function `len` (standard library)

comparison operators: lexicographical order (ASCII table)

```
>>> 'abc' + '-475'
'abc-475'
>>> len ('measures')
8
>>> 'abc'*3
'abcabcabc'
>>> 'a' < 'v'
True
>>>'7' > '9'
False
>>> 'leg' < 'wing'
False
>>> 'abracadabra'>'amber'
False
>>>'100' < '200'
True
>>> '100' < '2'
True
```

ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Useful intervals:

- [‘0’, ‘9’]
- [‘A’, ‘Z’]
- [‘a’, ‘z’]

builtins library

library: set of functions.
builtins library: set of built-in functions
already available

`int`, `float`, `str`, `bool`: conversion functions
`type`: type of a variable or value
`len`: number of elements of a sequence
`abs`: absolute value
`round`: rounds a real value
`max`, `min`: maximum and minimum of a set of values

```
>>> str (42.85)
'42.85'
>>> str (3*4)
'12'
>>> int (3*4.2)
12
>>> float (3*4)
12.0
>>> int('100')
100
>>> abs(-5.6)
5.6
>>> round(45.987623, 4)
45.9876
>>> max (4, 6, 5, 3)
6
>>> min (4, 6, 5, 3)
3
```

math library

`math` library: mathematical functions
we must **import** all libraries, except builtins
import statement, keyword: `import`
Two alternative sintaxes:

import library	function call
`import math`	`math.sin(x)`
`from math import sin`	`sin(x)`

```
>>> import math
>>> math.exp(4)
54.598150033144236
>>> math.log(54.598150033144236)
4.0
>>> math.sqrt(81)
9.0
>>> rad = math.radians(30)
>>> rad, math.sin(rad)
(0.5235987755982988, 0.4999999999999994)
```

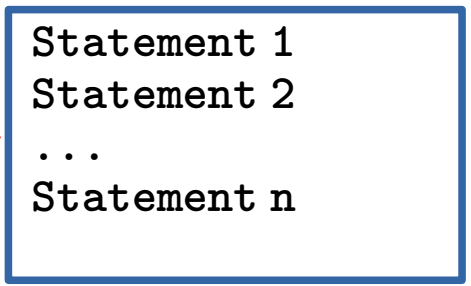
constants: `pi, e, ...`
`exp, log`: e^x, logarithm (e base)
`sqrt`: square root
`sin, cos, tan`: trigonometric functions (**radians**)
`degrees, radians`: conversion functions

Sequential composition

A program is a composition of statements.

3 main composition types:

- sequential:
- conditional
- repetitive



Statement i+1 is executed
after statement i, i =1..n-1

```
>>> a = 34   # line 1
>>> b = 58   # line 2
>>> a = a+b  # line 3
>>> b = a-b  # line 4
>>> a = a-b  # line 5
>>> a, b
(58, 34)
```

Comments: preceded by #

- intended for humans
- skipped by the Python interpreter

Program tracing

Tracing the value of the variables
along the program statements

automatic program tracing: Python tutor

Manual tracing (pencil and paper): table variables/statements

```
>>> a = 34    # line 1
>>> b = 58    # line 2
>>> a = a+b   # line 3
>>> b = a-b   # line 4
>>> a = a-b   # line 5
>>> a, b
(58, 34)
```

line	a	b
0	?	?
1	34	?
2	34	58
3	92	58
4	92	34
5	58	34

Code readability

- meaningful variable names
- **comments**:
  text following symbol # is not interpreted by Python

```
>>> a = 25
>>> b = 4
>>> c =  a/b
```

neither meaningful names nor comments: no idea of what intends to do this code

```
>>> mass = 25
>>> acceleration = 4
>>> force = mass/acceleration    # 2nd Newton's law
```

meaningful names and comments:

- purpose of the code easily guessed
- detection of errors

```
>>> mass = 25
>>> acceleration = 4
>>> force = mass*acceleration    # 2nd Newton's law
```

correct code

Multiple assignment

```
>>> a, b = 3, 4
```

1) right-hand side expressions are evaluated: values
2) left-hand side variable names are assigned these values

value swapping

```
>>> a, b = 3, 4
>>> b, a = a, b
>>> a, b
(4, 3)
```

no value swapping

```
>>> a, b = 3, 4
>>> b = a
>>> a = b
>>> a, b
(3, 3)
```

builtins library: `bool` and `eval`

`bool:` returns `True` except these cases

Unexpected result

`eval`: evaluates any expression represented as an string

```
>>> bool(0.0)
False
>>> bool(0)
False
>>> bool('')
False
```

```
>>> bool('True')
True
>>> bool('False')
True
```

```
>>> eval('True')
True
>>> eval('False')
False
>>> eval ('3*4')
12
>>> a = 5
>>> eval  ('a*2+6 < 20 and a < 10')
True
```

Input/Output

built-in functions `input` and `print`
`input`: gets input from the program user.
`print`: shows values, variables and expressions.

`input` function always returns a string
to obtain a value of another type
use conversion functions

```
>>> x = input ('Enter the value of x: ')
Enter the value of x: 25
>>> x
'25'
>>> x = float(input ('Enter the value of x: '))
Enter the value of x: 25
>>> x
25.0
>>> import math
>>> print('The square root of', x, 'is:', math.sqrt(x))
The square root of 25.0 is: 5.0
```