Simple Types

Type	Literals	Notes
integer	`123`	Any positive, negative or zero value; no limit on size
	`0`	
	`-12345`	
float	`3.14`	Real numbers; decimal or scientific notation; limits on scale and precision
	`6.02E23`	
boolean	`True`, `False`	Logical true/false quantities
string	`"Tuesday"`	Textual snippets comprising any number of characters; delimited by double quotes
	`""`	
	`"Luigi's"`	

Type conversions (int(), float() and str())

To integers: `int(3.14)`(truncates), `int("123")`; to reals: `float(3)`, `float("3.14")`; to strings: `str(123)`.

Int/Float Operators

Arithmetic operators: $+, -, *$ and parentheses with standard precedence rules; operator $**$ denotes exponentiation e.g. $2 ** 3 ==> 8$.

Op.	Example	Notes
//	`8 // 4 => 2`	Integer division: fractional part discarded
	`10 // 3 ==> 3`	
/	`8 / 4 ==> 2.0`	Real division: returns **float** i.e. fractional part retained
	`8 / 3 ==> 2.66`	
%	`8 % 4 ==> 0`	(Integer) remainder on division
	`8 % 3 ==> 2`	

Expressions with mixed integer and float operands yield float result, otherwise int result (apart from / operator).

Boolean expressions and conditions

Comparison operators: $==, !=, <, <=, >, >=$
Boolean operators: **not**, **and**, **or**

Variables and Assignments

Identifiers (variables etc.) comprise letters, digits and underscores, may not begin with a digit and are case sensitive.

Assignment

Python variables are untyped. No declarations! Use **type()** function to test a value's type.

```
                        #  x is      y is
                        #  ------    -------
x = 1 + 2 * 3           #  7
y = "seven"             #  7         "seven"
x = y                   #  "seven"   "seven"
x, y = 23, 101          #  23        101
x = x + 1               #  24        101
```

Other assignment operators: `+=`, `-=`, `*=` etc.

Input

```
mood = input("How are you?")
lucky_num = int(input("Enter lucky number:"))
```

Input prompt optional; strips end-of-line newline.

Print output

```
print(17)
print("seventeen")

# print items separated by spaces
print("one", "two", "three")

# use sep for alternative separator (here #)
print("one", "two", "three", sep = "#")

print("alpha", end = "") # suppress newline
print("beta")
```

Control Flow

Selection

```
if ⟨condition⟩:
    ⟨statements⟩

.....

if ⟨condition⟩:
    ⟨statements⟩
else:
    ⟨more statements⟩
```

```
if ⟨condition⟩:
    ⟨statements⟩
elif ⟨condition⟩:
    ⟨statements⟩
else:
    ⟨more statements⟩
```

e.g.

```
if age >= 65:
    print("Apply for bus pass")
```

Indentation

Indentation matters in Python. Indent statements of if/elif/else body four spaces (*not* tabs). Ditto for loop bodies and function bodies.

Repetition and Iteration

```
while ⟨condition⟩:
    ⟨statements⟩
```

```
for ⟨var⟩ in ⟨iterable⟩:
    ⟨statements⟩
```

e.g.

```
total = 0
n = 1
while n <= 100:
    total = total + n
    n = n + 1
```

```
total = 0
for n in range(101):
    total = total + n
```

range(n) generates values $0, 1, \cdots, n-1$; **range(n, m)** generates $n, \cdots, m-1$; opt. 3rd argument specifies step.

Functions

```
def shout():                    # function def'n.
    print("Hip Hip Horray!")

def plus_one(n):
    return n + 1                # return statement

reps = plus_one(2)              # function call
for n in range(reps):
    shout()                     # function call
```

Operator precedence: (lowest) **or**; **and**; **not**; **in**, $<, <=, >, >=, !=, ==$; $+, -$; $*, /, //, \%$; $**$ (highest)

Strings

String literals enclosed double-quotes ("). Blackslash (\) is the escape chararacter. Multiline strings delimited by three quote characters (""").

String Operations	
`s.strip(ch)`	Strip all ch chars from beginning and end of s; use `.lstrip()`, `.rstrip()` for one-sided version
`s.lower()`	Convert s to lowercase; also `.upper()` for uppercase
`s.center(n, ch)`	Centre s within string of length n, padding left/right with ch characters; `.ljust(n, ch)`, `.rjust(n, ch)` for left, right justifying versions
`s.replace(t, u)`	Replace every occurance of t with u (left to right)
`s.count(t)`	Count the number of occurences of t within s
`s.isnumeric()`	Determine if s consists entirely of numeric characters; also `.isalpha()`, `.isspace()` etc.; see book for list
`s.find(t)`	Index in s where t first occurs (−1 if nowhere); `.rfind()` for right to left search
`s.split(t)`	Split s into a list of substrings separated by t

String Formatting

Format characters: i (integer), f(real), s(string).

```
"Answer = %6i" % (42)      # field width 6
"Pi = %6.3f" % (3.14159) # width 6, matissa 3
"Sky is %s" % ("blue")     # width optional
```

Sequences (Strings, Lists and Tuples)

Python sequence types:

```
s = [2, 3, 5, 7, 11, 13] # list
s = ("one", 2, True)      # tuple (immutable)
s = "abcdefgh"            # string
```

Sequence Operations	
`len(s)`	Length of s
`s + t`	Concatenate s and t
`s * n`	Concatenate n copies of s
`s == t`	Return True is s and t are equal (item by item). Also <= etc. (lexigraphic order).
`s[i]`	i-th item of s

Splicing (any sequence type)

```
s = "abcdefgh"

print(s[2])       # ==> c
print(s[-3])      # ==> f (third from right)

print(s[2:5])     # ==> cde (items 2 to 5-1)
print(s[4:])      # ==> efgh (items 4 onwards)
print(s[:3])      # ==> abc  (items 0 to 3-1)

print(s[::2])     # ==> aceg (every 2nd item)
print(s[::-1])    # ==> hgfedcba (reversed)
```

Lists and Tuples (unmodifiable lists)

```
# x a mutable list, y an immutable tuple
x = ["a", "b", "c", "d", "e", "f", "g", "h"]
y = (2, 3, 5, 7, 11, 13)

print(x[2])    # ==> c
print(x[-3])   # ==> f

               # lists only for these
x[2] = "cee"   # set slot 2 to "cee"
x[-6] = "c"    # reverse the above; negative
               # indices work right to left

# iterate through list elements
for elt in x:
    print(elt)
```

List/Tuple Operations	
Name	**Note**
`a.append(v)`	Append v into end of list
`x.remove(v)`	Remove leftmost occurance of v
`x.insert(i, v)`	Splice v into list at index i
`x.index(v)`	Return index of leftmost occurance of v
`x.reverse()`	Reverse list order
`x.sort()`	Rearrange list into increasing order

List Comprehensions

```
# Generate list of first 20 odd nums
odds = [2*n + 1 for n in  range(20)]

# Generate list of primes less than 100
primes = [n for n in range(100) if is_prime(n)]
```

Dictionaries

```
animals = {"cow": "moo", "dog": "bark",\
       "duck": "quack"}

print(animals["duck"])  # ==> quack (lookup)

animals["dog"] = "woof" # (insertion/update)
print(animals["dog"])   # ==> woof (lookup)

# interate through keys/values
for beast in animals:
    print(beast, animals[beast])
```

File I/O

```
try:
   infile = open("poem.txt", "r")
   outfile = open("capped.txt", "w")
except IOError:
   print("Problems opeing file!")

for line in infile:
    all_caps = line.upper()
    outfile.write(all_caps)

infile.close()
outfile.close()
```

Operator precedence: (lowest) **or**; **and**; **not**; **in**, <, <=, >, >=, ! =, ==; +, −; ∗, /, //, %; ∗∗ (highest)