Scripting Languages

Python basics

Interpreter Session: `python`

- Direct conversation with python (>>>)

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Trying out commands, data and making simple experiments:
  - getting information on interpreter
  - making simple computations
  - using variables, text strings, lists, dictionaries, etc.
  - loading and examining modules
  - getting most-up-to-date help on everything

Numbers

- integers (unlimited length) or floats (64-bits)
  - also: complex (but you won't need this now)
- integer division (//), modulo (%), power (**)
- automatic type conversion in mixed cases

```
>>> 2 + 2
4
>>> 2 + 2.0
4.0
>>> 9 % 5
4
>>> 9 / 5
1.8
>>> 9 // 5
1
>>> 9.0 // 5
1.0
```

```
>>> 2**1024
1797693134862315907729305190789024733617976978942306
5727343008115773267580550096313270847732240753602112
0113879871393357658789768814416622492847430639474124
3777678934248654852763022196012460941194530829520850
0576883815068234246288147391311054082723716335051068
4586298239947245938479716304835356329624224137216
>>> 2.0**1024
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
>>>
```

Numbers

- binary and hex numbers with prefixes: 0x, 0b
- integer only

```
>>> 0b11111111
255
>>> 0xff
255
>>> 0xdeadbeef
3735928559
>>> 10 + 0x0A + 0b1010
30
>>> 0b11 * 0x11
51
>>> 2**0b1000
256
```

Logic Type

- Logic (boolean) type: True (1), False (0)
- Comparisons: ==, <, <=, >, >=
- Logic operators: and, or, not

```
>>> 1 == 1
True
>>> 2 > 3
False
>>> False - True
-1
>>> 1 == 2 or 2 <= 3
True
>>> not(False)
True
```

Type Check and Conversion

- check: type( )
- conversion: int( ), float( )
    - *fraction is lost in float → int conversion*

```
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> a = 9 / 5
>>> type(a)
<class 'float'>
>>> type(1j)
<class 'complex'>
>>>
```

```
>>> a = 3
>>> 1 + float(a)
4.0
>>> b = 1.5
>>> 1 + int(b)
2
>>> float(int(3.3))
3.0
>>> int(float(3.3))
3
```

Names

- Names (similar to variables, but different):
  - just labels, no type declaration needed
  - letter first + anything
  - forbidden: space,+, −, *, /, ", ', &, $, ^, #, @, ...
  - case sensitive

```
>>> a45 = 10
>>> 45a = 10
  File "<stdin>", line 1
    45a = 10
      ^
SyntaxError: invalid syntax

>>> _valid_Name = 13
>>>
```

Naming Conventions

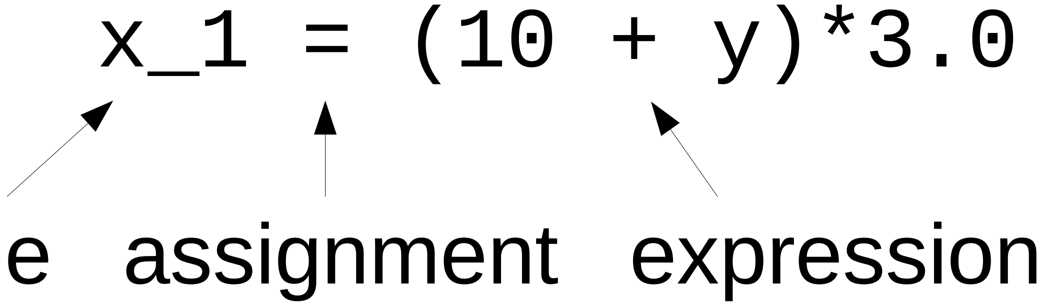
- lowercase
- lower_case_with_underscores
- UPPERCASE
- UPPER_CASE_WITH_UNDERSCORES
- CapitalizedWords
- mixedCase
- Capitalized_Words_With_Underscores

Camel vs Snake

- Camel Style
  - lowerCamelCase
  - UpperCamelCase
- Snake Style
  - lower_snake_case
  - Upper_Snake_Case
  - CAPITAL_SNAKE_CASE

- recommendations for Python
  - **UpperCamelCase** for class names,
  - **CAPITALIZED_WITH_UNDERSCORES** for constants,
  - **lower_snake_case** for other names.

Assignments



- Name(variable)
  - case-sensitive
- Assignment operator: single “=”
  - do not confuse with double '=='
- Expression
  - names, values, operators
  - operators priority: `**, -, */%,+-`
  - brackets

$-2^2 \neq (-2)^2$

```
>>> -2**2
-4
>>> (-2)**2
4
```

Augmented Assignments

- Assign to existing name with operation:

  x += 1   shorter/clearer than equivalent   x = x + 1

```
>>> x += 1
>>> x −= 2
>>> x *= 3
>>> x /= 4
>>> x %= 5
>>> x **= 6
```

Multiple Assignment

- Assign a single/many value to many variables

```
>>> i = j = k = 0.0
>>> x, y, z = 1, 2, 3
>>> x, y = y, x
```

  - the right-hand side of assignment is evaluated first

```
>>> a,b = 1, 2

>>> a,b = b, a+b

>>> a
2
>>> b
3
```

⇔ !

```
>>> a, b = 1, 2

>>> a = b
>>> b = a+b

>>> a
2
>>> b
4
```

Exercise

Temperature conversion:

- write a 4-line program in python
- use names for Celsius, Kelvin, Fahrenheit
- find the proper formula yourself

```
c = 7
k = ...
f = ...
print(c, k, f)
```

Text Strings

- String: a sequence of any characters
- Strings are surrounded by matched:
  - single quotes
  - double quotes
  - triple quotes (3-single or 3-double):

```
>>> s = 'This is a valid string'
>>> s = "This is a valid string"
>>> s = '''This is a valid string'''
>>> s = """This is a valid string"""
```

Text Strings

- Double quotes are allowed inside single-quoted strings
- Single quotes are allowed inside double-quoted strings
- Almost anything is allowed inside triple-quoted strings (also new-line breaks)

```
>>> s = ' This is a valid string with " inside '
>>> s = " This is a valid string with ' inside "
>>> s = '''This is a valid string
... with ' and " and line breaks inside'''
```

String Operations

- For full list run: `help(str)`
  - `len(s)`
  - `s1 + s2`
  - `s.isdigit()`
  - `s.isalpha()`
  - `s.upper()`
  - `s.lower()`
  - and others

```
>>> a = "Hello"
>>> type(a)
<class 'str'>
>>> len(a)
5
>>> a.isdigit()
False
>>> a.isalpha()
True
>>> a.upper()+a.lower()
'HELLOhello'
```

String Slicing

- Numbered sequence
  - string `s="Hello"`
  - position `01234`
- Substrings: [ ]
  - first char at position 0
  - last char at position -1
  - access range with ":"
    - 0:n means n chars,
      - from 0 to n-1 position

```
>>> x = "Hello world"
>>> x[0]
'H'
>>> x[0:4]
'Hell'
>>> x[:4]
'Hell'
>>> x[:]
'Hello world'
>>> x[-1]
'd'
>>> x[3:-1]
'lo worl'
>>> x[-4:-1]
'orl'
>>> x[-4:]
'orld'
```

String Formatting

- Gluing together strings and numbers
  - `string.format("pattern",args)`
  - `{n}` for argument placement and position
  - `d`, `f`, `s` for int, float and string types

```
>>> a = "I'm {0} and I'm {1}".format('John',20)
>>> print(a)
I'm John and I'm 20

>>> print("T={0:2f}C is {1:+.2f}K".format(c, k))
T= 1C is +274.15K
```

https://docs.python.org/3/library/string.html#format-specification-mini-language

String Formatting (2)

- f-strings (Python ver. 3.6+)
  - `f"any {name1} text {name2} inside"`
  - `{name}` – placeholder with variable name
  - `{name:format}` – with formatting options

```
>>> person = 'John'
>>> age = 20
>>> print( f"I'm {person} and I'm {age}" )

I'm John and I'm 20

>>> print(f"T={c:.2f}C is {f:+.2f}F {k:+.2f}K")

T=37.77C is +99.99F +310.92K
```

Comments

- Comments start with hash ‘#’ character
- Everything after ‘#’ is ignored till the end of line
- Use meaningful comments in your code !

```
# width
w = 10

# length
l = 20

a = w * l   # area
```

```
# bad
x = x + 1   # increment x

# good
x = x + 1   # add bias to x
```

Decisions → if-elif-else

- conditions are bool-type followed by colon ‘:’
- body instructions are indented (use 4-spaces)

```
if cond:
    instr
```

```
if cond:
    instr1
else:
    instr2
```

```
if cond1:
    instr1
elif cond2:
    instr2
else:
    instr3
```

```
if a>0:
    print("positive")
else:
    print("not positive")
```

```
if a>0:
    print("positive")
elif a<0:
    print("negative")
else:
    print("zero")
```

Conditional Loop → while

- condition bool-type in the first line
  - in Python there is no loop with condition at the end (nothing like *do-while* exists)
- body indented

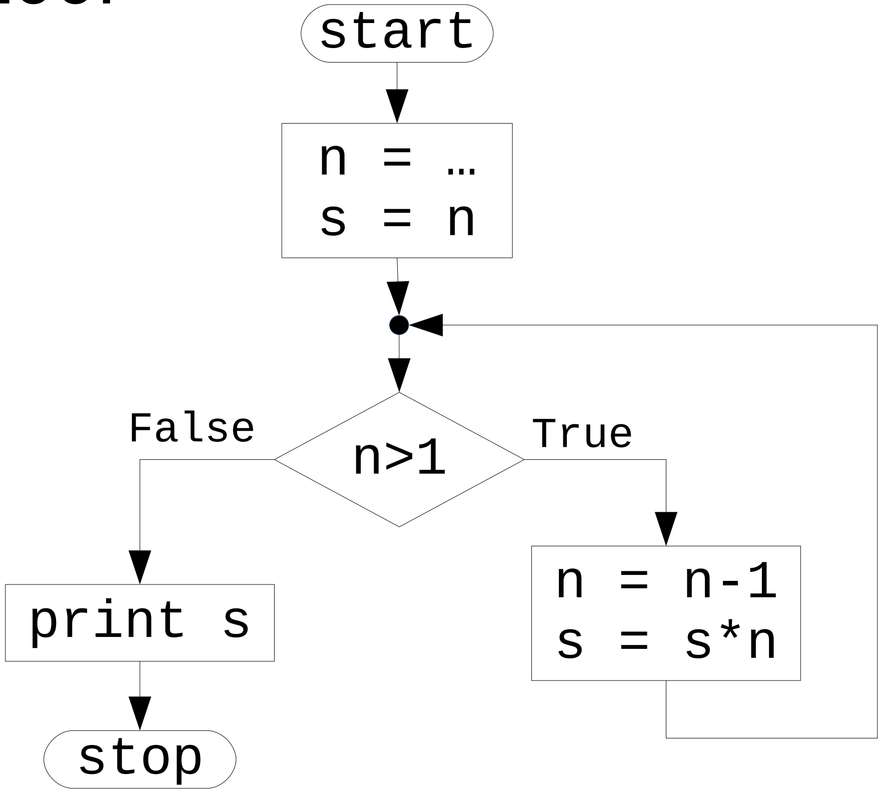
```
while cond :
    instr
```

```
while i < j:
    i = i+1
    j = j-1
```

Exercise: Factorial

Factorial n! = n·(n-1)·...2·1

- how far you can go with value of n?
- what is the best data type for result ‘s’?
- how many digits has 100!



Loop control → break, continue

- break → immediate escape from the loop
- continue → immediate start of the next cycle

```
while cond1:
    if cond2:
        break
    instr
```

```
while cond1:
    if cond2:
        continue
    instr
```

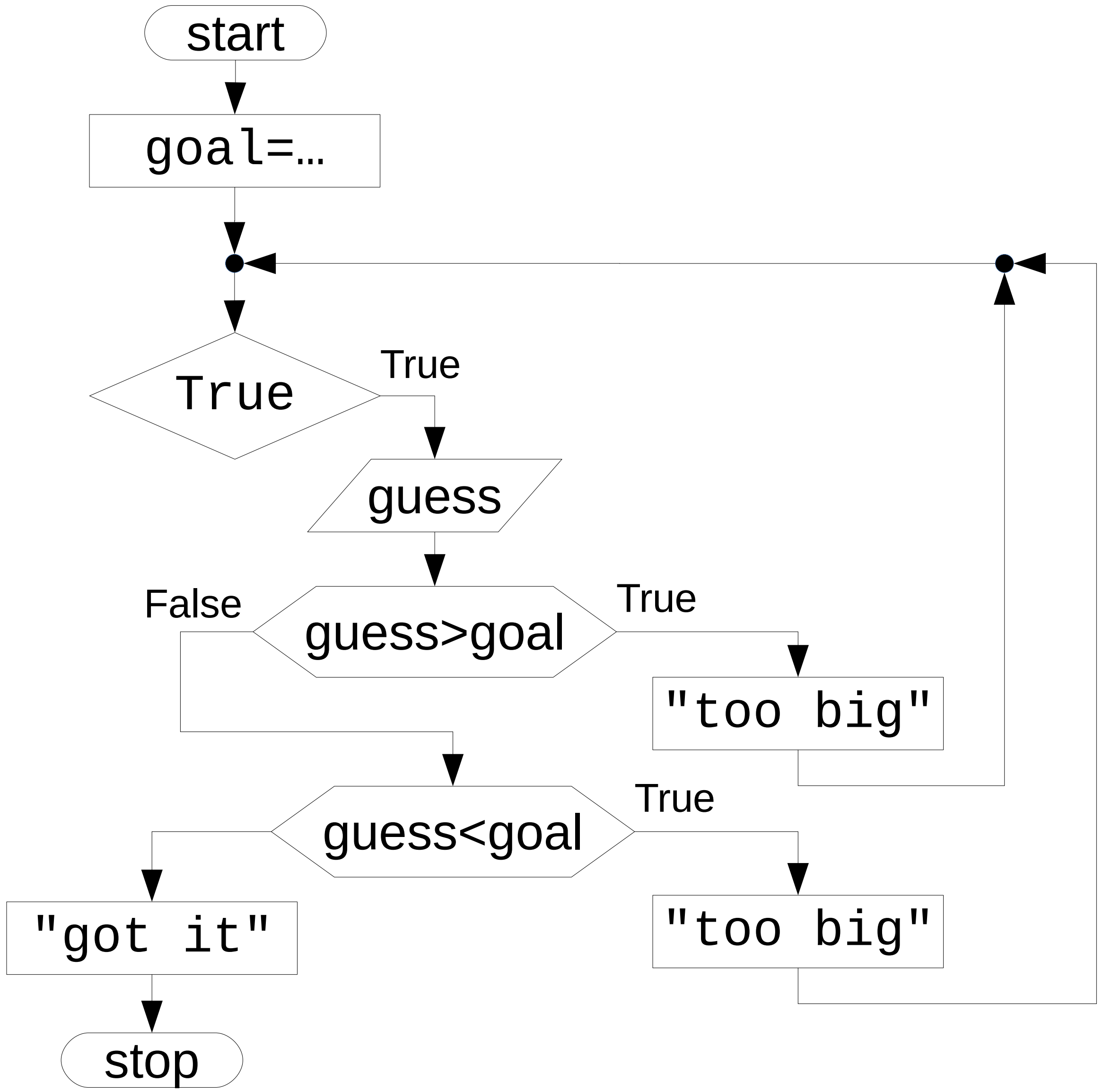
```
a = 0
while a < 10:
    a = a + 1
    if a**4 > 1000:
        break
    print(a, a**4)
```

```
a = 11
while a > -10:
    a = a - 1
    if a==0:
        continue
    print(a, 1.0/a)
```

Exercise: Guess-It-Game

```
goal = …

while True:
    read guess
    if …
        print …
    elif …
        print …
    else
        print …
        break
```



Getting Input for CLI

- s = input("prompt")
  - print a "prompt" message and return input as text

    s.isdigit() → make sure there are digits only

    n = int(s) → convert to integer

```
while True:
    s = input("Number -> ")
    if not s.isdigit():
        continue
    n = int(s)
    …
```

Random values

- Add a library (module) and use function

  ```
  import random
  random.function()
  ```

- Read documentation
  - Python 3 Standard Library
    - 9. Numeric and Mathematical Modules
      - 9.6. random — Generate pseudo-random numbers
        - random.randint(a, b)
          Return a random integer N such that a <= N <= b

```
import random
…
guess = random.randint(1,100)
```

Math functions

- Add a library (module) to your program

  ```
  import math
  ```

- Read documentation
  - Python 3 Standard Library
    - 9. Numeric and Mathematical Modules
      - 9.2. math — Mathematical functions

```
import math
…
x = math.sqrt(math.cos(2*math.pi/T)**2 -1)
```