#python notes *2024 03 14*

thonny, pythontutor, https://book.pythontips.com

; in C, : in Python

```
if __name__ == "__main__":
   print("hello world")
```

id(): mutabili e non

```
x = [0]
print(id(x))
x.append(1)
print(id(x))
x = x + [2]
print(id(x))
```

Curiosità:

```
x=256
y = x + 1
y = y - 1
print(x, id(x), id(y), x is y)
x=257
y = x + 1
y = y - 1
print(x, id(x), id(y), x is y)
```

x="ciao" in C e x="ciao" in Python

ovvero scatola contro targhetta

None

non esistono funzioni che non ritornano nulla, ma funzioni che restituiscono nulla

Numeri int float complex:

int a precisione illimitata, float IEEE double

tipi iterabili: string tuple set list dict

- “ciao” “”
- (1,2,3) 1,2,3 (1,) ()
- {1,2,3} set()
- [1,2,3] []
- {1:10, 2:20, 3:30} {}

assegnamenti tra iterabili

-

loop: for, while (con caso else)

pass

with

```
with open(filename, "r") as f:
  x = f.read()
print(x)
```

espressione if

```
def veritas(x):
   return 'vero' if x else 'falso'
```

Argomenti di funzione:

- posizionali
- con valore di default
- espressi per nome
- funzioni variadiche con * e **

iteratori

```
class itenumerate:
  def __init__(self, seq):
    self.seq = seq
  def __iter__(self):
    self.i = 0
    return self
  def __next__(self):
    if self.i >= len(self.seq):
      raise StopIteration
    this = self.i
    self.i += 1
    return this, self.seq[this]

for num, val in itenumerate(['a','b','c']):
  print(num,val)
```

Generatori

- funzioni che usano yield invece di return
- ogni chiamata costruisce e ritorna un iteratore (funzione con metodo next utile per i loop `for`)

- se la funzione termina emette 'StopIteration'

```
def myenumerate(seq):   # la enumerate è built-in
  n = 0
  for item in seq:
    yield n, item
    n += 1
for num, val in myenumerate(['a','b','c']):
  print(num,val)
```

```
def fibonacci():
  i = j = 1
  while True:
    r, i, j = i, j, i + j
    yield r
for rabbits in fibonacci():
  print(rabbits, '', end='')
  if rabbits > 100: break
print()
```

Decoratori

```
@<decorator>
def <name> etc, etc
```

è come

```
def <name> etc, etc
<name> = <decorator>(<name>)
```

trace.py:

```
def trace(f):
  f.indent = 0
  def strtuple(x):
    return "("+str(x[0])+")" if len(x)==1 else str(x)
  def g(*x):
    print('| ' * f.indent + '/-- ', f.__name__, strtuple(x), sep='')
    f.indent += 1
    value = f(*x)
    f.indent -= 1
    print('| ' * f.indent + '\-- ', 'return', repr(value))
    return value
  return g

def memoize(f):
  cache = {}
  def g(*x):
    if x not in cache:
```

```
      cache[x] = f(*x)
    return cache[x]
  return g
```

main.py:

```
#!/usr/bin/env python3
import sys
import trace

# try to uncomment the following statements:
#@trace.trace
#@trace.memoize
def fib(i):
  return 0 if i<=0 else 1 if i==1 else fib(i-1)+fib(i-2)

if __name__=="__main__":
  for i in range(int(sys.argv[1])):
    print(i,fib(i))
```

Overload degli operatori

Metodi speciali che iniziano e finiscono con il doppio underscore

```
__new__ __init__ __del__    # init/final.
__repr__ __str__ __int__    # conversions
__lt__ __gt__ __eq__ ...    # comparisons
__add__ __sub__ __mul__ ... # arithmetic
__call__ __hash__ __nonzero_ ...
__getattr__ __setattr__ __delattr__
__getitem__ __setitem__ __delitem__
__len__ __iter__ __contains__
```

Notare: la funzione dir.

Comprehension

```
>>> (2*x for x in range(4))
<generator object <genexpr> at 0x7fb7e4672260>
>>> [2*x for x in range(4)]
[0, 2, 4, 6]
>>> {2*x for x in range(4)}
{0, 2, 4, 6}
>>> {x : 2*x for x in range(4)}
{0: 0, 1: 2, 2: 4, 3: 6}
```

scrivere *librerie*: moduli e package

esempio1, modulo: file come libreria m.py: “‘python#!/usr/bin/env python3 import lib

if **name** == “**main**”: lib.hw()

lib.py:
```python

#!/usr/bin/env python3
def hw():
        print("hello world")

if __name__ == "__main__":
        print("test code")
```

esempio2, package m.py:

```python
#!/usr/bin/env python3
import lib

if __name__ == "__main__":
        lib.hw()
```

lib/___init___.py:

```python
#!/usr/bin/env python3
from .hw import hw
x=42
```

lib/hw.py:

```python
#!/usr/bin/env python3
def hw():
        print("hello world")

if __name__ == "__main__":
        print("test code")
```