Titolo del corso o seminario:	Programmare con Python3.3 per chi non ha mai programmato: partendo da zero Nota bene: questo programma è stato pensato come spiegazione del testo Non-Programmer's_Tutorial_for_Python_3: La sua struttura quindi segue il testo.
Relatore/relatori	Incontro 1:dott./prof./eccell.issimo..... incontro 2:dott./prof./eccell.issimo..... incontro 3:dott./prof./eccell.issimo..... (memento preparare i dati relativi alla costruzione del grafico al termine del terzo incontro)
prerequisiti:	a) conoscenza basilare di un sistema GNU/Linux Debian b) saper usare benino (mouse e) la tastiera di un pc c) conoscenza minimo-decente dell'inglese scritto d) esser incuriositi verso il mondo della programmazione
obbiettivi:	Saper programmare in Python3.3 Il corso fornirà le conoscenze di base del linguaggio con l'eccezione di classi ed oggetti.
durata:	il corso è strutturato in tre lezioni di circa tre ore l'una, per un totale di nove ore
sede:	il corso sarà tenuto presso la sede del FSUG Padova http://www.fsugpadova.org
orario:	salvo complicanze dalle 9 alle 12 circa di sabato 30, 44, 58 febbraio 2013
costi:	L'iscrizione al corso costa 25,00 euro. L'iscrizione al FSUG Padova è obbligatoria e costa attualmente 25,00 € all'anno. L'iscrizione al corso da diritto alla sola frequenza dello stesso.

		Incontro n°	1	Programmare con Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
apertura		5	0-5	
		8	5-13	Burocrazie e presentazione partecipanti
		6	13-19	caratteristiche, in breve, di Python
		2	19-21	domande?
		4	21-25	caratteristiche di questo corso
		2	25-27	domande?
		6	27-33	come si installa Python3.3 (solo i link ai siti)
documentazione del corso		0	33-33	http:/ / www. python. org/ download/
		3	33-36	dove si scarica la documentazione per questo corso (per chi non l'ha già fatto!!!)
IL TESTO DI RIFERIMENTO		2	36-38	http://en.wikibooks.org/wiki/Non-Programmer's_Tutorial_for_Python_3/Print_version
		4	38-42	in italiano me la chiedete, che non ricordo dove l'ho trovata...

		Incontro n°	1	Programmare con Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
le mie prime righe di codice		4	42-46	Python3.3 >>> print (“ciao mondo!”) >>> 2+2 >>> 4
		4	46-50	spiegazione
il mio primo programma		4	50-54	Save “ciao.py” .!. run Module
		4	54-58	spiegazione
		5	58-63	stessa cosa con un editor (idle3 ?) e poi python3.3 nomeprogramma.py
		4	63-67	o in una shell ma dando i permessi con chmod e #!/usr/bin/env python3.3
		3	67-70	Documentazione:
		0	70-70	http://www.python.org/doc/3.2/tutorial/index.html
		0	70-70	http://www.python.org/doc/3.2/library/index.html
		0	70-70	http://www.python.org/doc/3.2/reference/index.html
per aiuto on-line		4	70-74	liste:
		1	74-75	Imparare: http://mail.python.org/mailman/listinfo/tutor
		0	75-75	help: http://www.python.org/community/lists/#python-help
		1	75-76	usergroup: http://wiki.python.org/moin/LocalUserGroups
		0	76-76	discussioni: http://groups.google.com/group/comp.lang.python/
		1	76-77	italia: http://www.python.it/
		0	77-77	routa e fuoco sono già stati inventati: RTFM
		5	77-82	printiamo un poco..
le funzioni		4	82-86	e spiegare come funziona la funzione PRINT con i suoi argomenti ed un interprete
		1	86-87	ed una stringa “...”
		12	87-99	pausa relax
		0	99-99	chiamare una funzione → fn() :-) abbiamo fatto una chiamata a funzione
		2	99-101	una chiamata a funzione è uno statement (istruzione)
print e le operazioni..		4	101-105	Print (“3 + 2 =” , 3 + 2) come calcolatore
		2	105-107	Elevamento a potenza (simbolo **) 5** 2 == 25 (== vuol dire è eguale!)
		1	107-108	Moltiplicazione (*) 2 * 3 == 6

		Incontro n°	1	Programmare con Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
		1	108-109	Division (/) 14 / 3 == 4.666666666666667
		0	109-109	Integer Division (//) 14 // 3 == 4
		1	109-110	Resto (divisione modulo) (%) 14 % 3 == 2
		0	110-110	Somma (+) 1 + 2 == 3
		3	110-113	Sottrazione (-) 4 - 3 == 1
		12	113-125	ordine delle operazioni
		3	125-128	Domande?
i commenti!		2	128-130	i commenti # (ricordare i commenti ### ?)
i nomi		0	130-130	Identifiers → nomi che identificano un oggetto in python
		1	130-131	regole dei nomi:
		0	131-131	debbono cominciare o con una lettera o con il segno _
		1	131-132	poi si può mettere quel che si vuole fuorché
		1	132-133	'@ $ %
		2	133-135	minuscole e maiuscole NON sono la stessa cosa
		0	135-135	Cavolo <> da cavolo
		2	135-137	Però ci sono delle abitudini consolidare:
		1	137-138	si usa iniziare con una maiuscola SOLO il nome delle classi
		1	138-139	un nome che inizia con il segno _ (la sottolineatura) significa che il nome è inteso come privato (vedremo poi...)
parole riservate		2	139-141	e con due __ che lo è davvero!!
		1	141-142	e se inizia e finisce con due: __nome__ sarà usato in futuro dal linguaggio
		2	142-144	Ci sono una serie di parole che sono di uso riservato, le istruzioni del linguaggio
il linguaggio			144-144	False, None, True,
			144-144	and, as, assert,
			144-144	break
			144-144	class, continue
			144-144	def, del
			144-144	elif, else, except,
			144-144	finally, for, from,
			144-144	global,
			144-144	if, import, in, is,
			144-144	lambda,
			144-144	nonlocal, not,
			144-144	or,
			144-144	pass,
			144-144	raise, return,

		Incontro n°	1	Programmare con Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
			144-144	try,
			144-144	while, with,
			144-144	yield,
linguaggio ed inizio riga			144-144	l'uso degli spazi: Python è un linguaggio posizionale
			144-144	ed obbliga ad essere ordinati!
i tipi di variabile			144-144	tipi base sono numeri e stringhe (di caratteri)
operatore di assegnazione		6	144-150	gigi = True
if			150-150	L'istruzione di controllo di flusso if (ed i blocchi di codice: if gigi: # questo blocco funziona
			150-150	print(“vero”)
			150-150	else:
			150-150	print(“falso”)
		6	150-156	if False: # questo blocco non funziona
			156-156	print(“vero”)
			156-156	else:
			156-156	print(“falso”)
			156-156	print(“non funziono”)
			156-156	# così spieghi if ed i blocchi di istruzione!
se si vuole ...		5	156-161	lanciare un programma in automatico: #!/usr/bin/env python
			161-161	./nomeprog.py CHMOD prima
			161-161	alex = input(“che cavolo dici?”)
			161-161	print alex
			161-161	alex
operatori di confronto		4	161-165	'= operatore di assegnazione non è == operatore di confronto
incremento...		2	165-167	'a = 0; a = a+1 etc
			167-167	tornando alle variabili:
il tipo float		3	167-170	Pippo = float(input(“Scrivi un numero:”))
			170-170	print(“float: “, type(pippo))
il tipo int		2	170-172	Pippo = int(input(“Scrivi un numero intero:”))
			172-172	print(“intero: “, type(pippo))
il tipo stringa		3	172-175	Pippo = “questo è un testo”
			175-175	print(pippo * 2)
		2	175-177	print (pippo + pippo + “ “ + pippo)
		2	177-179	print (pippo – pippo)
		12	179-191	domande ed alla prossima volta

		Incontro n° 2		Programmare in Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
apertura		8	0-8	
i cicli		4	8-12	'a= 0
< e >		6	12-18	'while a < 10
		1	18-19	print('qui a vale ',a)
		3	19-22	a = a + 1
		0	22-22	print('invece qui a vale ',a)
		3	22-25	print('ho finito !!!')
		1	25-26	's = 0
		0	26-26	'a= 1 # !!!
Diverso !=		3	26-29	'while a != 0 # chi si accorge come finisce il ciclo?
		0	29-29	print('la somma attuale è ',s)
anche a = 3.0		1	29-30	a = float(input('scrivi un numero e batti invio! '))
		1	30-31	s = s +a
		3	31-34	print('ho finito ! Il totale è ',s)
end='		0	34-34	a = 0 # fibonacci
		0	34-34	b = 1
		0	34-34	'conta = 0
		1	34-35	'fino_a = 20
		3	35-38	while conta < fino_a
		0	38-38	conta = conta + 1
		0	38-38	print(a, end= ')
		0	38-38	adiprima = a
		0	38-38	a = b
		0	38-38	b = adiprima + b
		1	38-39	Print() # accapo
			39-39	name = input("What is your UserName: ")
			39-39	password = input("What is your Password: ")
			39-39	print("To unlock your computer type unlock.")
uso di None		7	39-46	command = None
			46-46	input1 = None
			46-46	input2 = None
fermi successivi non nidificati			46-46	while command != "unlock":
			46-46	command = input("What is your command: ")
			46-46	while input1 != name:
			46-46	input1 = input("What is your username: ")
			46-46	while input2 != password:
			46-46	input2 = input("What is your password: ")
			46-46	print("Welcome back to your system!")
			46-46	name = input('Set name: ')
			46-46	password = input('Set password: ')
gira per sempre		7	46-53	while 1 == 1:
			53-53	nameguess=""
			53-53	passwordguess=""
			53-53	key=""

	Durata reale	Incontro n° 2 Durata stimata	Tempi in minuti:	Programmare in Python3.3 partendo da zero
				while (nameguess != name) or (passwordguess !
si noti l'uso di 'or'			53-53	= password):
			53-53	nameguess = input('Name? ')
			53-53	passwordguess = input('Password? ')
			53-53	print("Welcome,", name, ". Type lock to lock.")
			53-53	while key != "lock":
			53-53	key = input("")
torniamo ad if		5	53-58	n = int(input("Number? "))
			58-58	if n < 0:
			58-58	print("The absolute value of", n, "is", -n)
			58-58	else:
			58-58	print("The absolute value of", n, "is", n)
< > | <= | >= | == | !=		3	58-61	comparatori
51 + 51 > 100				
True			61-61	a = 0
inutile qui ma comodo spiegarlo		2	61-63	'while -1 < a < 10:
			63-63	a = a + 1
			63-63	if a > 5:
			63-63	print(a, ">", 5)
uso di endif elif		5	63-68	elif a <= 3:
			68-68	print(a, "<=", 3)
			68-68	else:
			68-68	print("Neither test was true")
Debugging		4	68-72	vi ci divertirete molto..
uso di def nomefunzione(...) :		5	72-77	def ciao(a_chi):
			77-77	print(“ciao davvero “,a_chi)
			77-77	boia = input('ma chi sei?')
			77-77	ciao(boia)
variabili globali e locali		10	77-87	a = 4
			87-87	a_var = 10
			87-87	b_var = 15
			87-87	e_var = 25
			87-87	def a_func(a_var):
			87-87	print("in a_func a_var= ", a_var)
			87-87	b_var = 100 + a_var
			87-87	d_var = 2 * a_var
			87-87	print("in a_func b_var= ", b_var)
			87-87	print("in a_func d_var= ", d_var)
			87-87	print("in a_func e_var= ", e_var)
			87-87	return b_var + 10
			87-87	
			87-87	c_var = a_func(b_var)

		Incontro n° 2		Programmare in Python3.3 partendo da zero
	Durata reale	Durata stimata	Tempi in minuti:	
			87-87	
			87-87	print("a_var “, a_var)
			87-87	print("b_var “, b_var)
			87-87	print("c_var “, c_var)
			87-87	print("d_var “, d_var)
		6	87-93	Trovare l'errore???
			93-93	The variables b_var and d_var are local variables since they appear on the left of an equals sign in
			93-93	the function
			93-93	Another thing to notice is the NameError that happens at the end. This appears since the variable d_var no
			93-93	longer exists since a_func finished. All the local variables are deleted when the function exits. If you want to get
			93-93	something from a function, then you will have to use return something.
		12	93-105	pausa da stress
funzioni ricorsive		8	105-113	def mult(a, b):
			113-113	if b == 0:
			113-113	return 0
			113-113	rest = mult(a, b - 1)
			113-113	value = a + rest
			113-113	return value
			113-113	print("3 * 2 = ", mult(3, 2))
			113-113	Probably the most intuitive definition of recursion is:
			113-113	Recursion
			113-113	If you still don't get it, see recursion.
liste		4	113-117	'mesi = ['gen', febra',3,4,'maggio,6]
			117-117	o anche liste di liste
		4	117-121	'di_tutto = [1,2,mesi,3,'sandro','matti']
			121-121	caratteristiche di sintassi.
riferimento agli elementi di una lista			121-121	varabile_a = nomelista[numero_elemento]
aggiungi			121-121	nomelista.append(nuovo_elemento)
numero elementi			121-121	num_elem = len(nomelista)
dal numero al numero..			121-121	range(start,stop,step) range(start,stop) range(stop)
iteratore			121-121	#'infa scorrere tutti gli elementi di un oggetto
ciclo for		5	121-126	for c in range(len(nomelista)):
iteratore			126-126	print(nomelista(c)) #c è l'indice
		3	126-129	for c in nomelista
indice			129-129	print(c) # c è il valore
cancella		2	129-131	posto_in_lista = nomelista.index(valore_elemento)

	Durata reale	Durata stimata	Tempi in minuti:	
trova anche per nome		3	131-134	del nomelista[posto_in_lista]
		2	134-136	if elemento in nomelista:
ordina			136-136	print(nomelista[elemento])
		2	136-138	nomelista.sort()
			138-138	alcuni metodi delle liste:
				| L.append(object) -> None -- append object
			138-138	to end
		1	138-139	| L.clear() -> None -- remove all items from L
			139-139	| L.copy() -> list -- a shallow copy of L
				| L.count(value) -> integer -- return number of
		3	139-142	occurrences of value
				| L.extend(iterable) -> None -- extend list by
		2	142-144	appending elements from the iterable
				| L.index(value, [start, [stop]]) -> integer --
		2	144-146	return first index of value.
			146-146	| Raises ValueError if the value is not present.
				| L.insert(index, object) -- insert object before
		2	146-148	index
				| L.pop([index]) -> item -- remove and return
			148-148	il valore che era ad index (default last).
				| Raises IndexError if list is empty or index is
			148-148	out of range.
				| L.remove(value) -> None -- remove first
		2	148-150	occurrence of value.
			150-150	| Raises ValueError if the value is not present.
		2	150-152	| L.reverse() -- reverse *IN PLACE*
				| L.sort(key=None, reverse=False) -> None --
		2	152-154	stable sort *IN PLACE*
		2	154-156	listavouta = list()
		3	156-159	listapiena = list(range(10))
		1	159-160	print(listapiena)
			160-160	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
un'excursus sui booleani		6	160-166	'v = True
				f = False # provare a scrivere sul terminale
			166-166	Python:
			166-166	v and v
			166-166	v and f
			166-166	f and v
			166-166	f and f
		6	166-172	e ripetere con or al posto di and
			172-172	poi a = 'pippob = 'minni e ripetere
		3	172-175	provare:
		6	175-181	'a = 'pippo'
		6	181-187	'b = 'minni'
			187-187	('aor 'b')
			187-187	('bor 'a')

	Incontro n° 2		Programmare in Python3.3 partendo da zero
Durata reale	Durata stimata	Tempi in minuti:	
		187-187	('aand 'b')
		187-187	('band 'a')
	10	187-197	saluti ed alla prossima con i dizionari

	Durata reale	Incontro numero 3		Programmare in Python3.3 partendo da zero
		Durata	Tempi in minuti:	
apertura		8	0-8	
il tipo dict		4	8-12	dizionario = {} e creo un dizionario vuoto
		6	12-18	sostanzialmente sono accoppiate nomi/valore
		1	18-19	'dizionario['beppi'] = 123,456
		5	19-24	'dizionario['alex']=[123,'figo',[1,2,3,4,5]]
l'istruzione type()		0	24-24	da provare
		3	24-27	si provino le funzioni e le ricerche che abbiamo visto per le liste
da console provare		15	27-42	sostituendo dizionari a liste
help(dict)		0	42-42	
ed help(list)		3	42-45	metodi
		1	45-46	| D.clear() -> None. Remove all items from D.
		2	46-48	| D.copy() -> a shallow copy of D
		2	48-50	| dict.fromkeys(S[,v]) -> New dict with keys from S and values equal to v.
		1	50-51	| v defaults to None.
		2	51-53	| D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.
		2	53-55	| D.items() -> a set-like object providing a view on D's items
		0	55-55	| D.keys() -> a set-like object providing a view on D's keys
		2	55-57	| D.pop(k[,d]) -> v, remove specified key and return the corresponding value or d
		1	57-58	| If key is not found, d is returned if given, otherwise KeyError is raised
		1	58-59	| D.popitem() -> (k, v), remove and return some (key, value) pair as a
tuple		4	59-63	| 2-tuple; but raise KeyError if D is empty.
		0	63-63	| D.setdefault(k[,d]) -> D.get(k,d), also set D[k]=d if k not in D
		3	63-66	| D.update([E,]**F) -> None. Update D from dict/iterable E and F.

	Durata reale	Incontro numero 3		Programmare in Python3.3 partendo da zero
		Durata	Tempi in minuti:	
		2	66-68	| If E present and has a .keys() method, does: for k in E: D[k] = E[k]
		2	68-70	| If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v
		3	70-73	| In either case, this is followed by: for k in F: D[k] = F[k]
		3	73-76	| D.values() -> an object providing a view on D's values
		1	76-77	| dict() -> new empty dictionary
		7	77-84	| dict(mapping) -> new dictionary initialized from a mapping object's
			84-84	| (key, value) pairs
		5	84-89	| dict(iterable) -> new dictionary initialized as if via:
			89-89	| d = {}
			89-89	| for k, v in iterable:
			89-89	| d[k] = v
		5	89-94	| dict(**kwargs) -> new dictionary initialized with the name=value pairs
			94-94	| in the keyword argument list. For example: dict(one=1, two=2)
usare i moduli, importato tutto			94-94	import calendar
			94-94	anno = int(input('un anno?))
			94-94	calendar.prcal(anno)
importato solo un metodo		3	94-97	Oppure from calendar import prcal
		5	97-102	provare a chiamare help(calendar)
		5	102-107	from time import time, ctime
			107-107	'prev_time =””
			107-107	'while True: #per uscire break !!
			107-107	thetime = ctime(time())
			107-107	if prev_time = thetime
			107-107	Oppure Ctrl C per interrompere
Slicing		5	107-112	lista = ['a', 'b', 'c', 'd', 'e']
			112-112	altra_lista = lista[2:4]
			112-112	Dal 2 compreso al 4 esclude
		4	112-116	come le liste gli indici funzionano anche al negativo
			116-116	ma una lista incorporata è solo un indice. Provare
			116-116	'a = list()

	Durata reale	Incontro numero 3		Programmare in Python3.3 partendo da zero
		Durata	Tempi in minuti:	
			116-116	'a = [1,2,3]
			116-116	B = list()
		3	116-119	B= ['a','b',a,5]
			119-119	print(b)
		2	119-121	C = B
			121-121	print(c)
			121-121	B[2]='cavolo fai?'
			121-121	print(c)
		5	121-126	C = B[:]
			126-126	print(c)
			126-126	B[2]='cavolo fai?'
			126-126	print(c)
		4	126-130	Page 72
passare i riferimenti		5	130-135	fate attenzione:
e passare i valori			135-135	a=[1,2,3,4,5]
			135-135	b= a
			135-135	print(b)
		8	135-143	A[1]=9
			143-143	print(b)
con i tipi base non accade			143-143	Riprovate ma con b=a[:]
			143-143	oppure con a = b.copy()
			143-143	se v'é in mezzo un'operazione... a = b * 2 # b è una lista...
			143-143	ma se nella lista da copiare vi sono riferimenti
			143-143	ad altre liste allora:
			143-143	import copy # oppure import deepcopy from copy
copiare tutto		2	143-145	b =copy.deepcopy(a)
		6	145-151	ord(carattere), chr(int), repr(numero), int(testo o float),float(testo o int),eval(testo)
		5	151-156	split() # è un metodo di str
			156-156	“questo, è, un testo, complesso”.split(',', 2)
			156-156	e provate help(str.split)
			156-156	li slicing funziona anche sulle stringhe..
scrivere su un file		6	156-162	with open('nomefile.txt,'wt') as nomeIOWr:
			162-162	nomeIOWr.write(testo)
			162-162	with open('nomefile.txt','rt') as nomeOIWr:
			162-162	testo = NomeOIWr.read()

	Durata reale	Incontro numero 3		Programmare in Python3.3 partendo da zero
		Durata	Tempi in minuti:	
			162-162	a cosa serve with
		6	162-168	per dettagli: help(close)
			168-168	per chiamare un programma:
			168-168	'import subprocess
		8	168-176	a = subprocess.call(['vim', '/home/alex/prova.txt'])
		8	176-184	try: except
		8	185-192	saluti ed alla prossima

Contributors:

Alessandro Medici, Franco Vecchiato